Master Thesis

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Neural Network Based Eyetracking System

Bc. Adrián Pitoňák

Supervisor: Ing. Jaromír Doležal, Ph.D.
Field of study: Open Informatics
May 2022

MASTER‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Pitoňák Adrián**	Personal ID number:	**474380**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Cybernetics**		
Study program:	**Open Informatics**		
Specialisation:	**Computer Vision and Image Processing**		

II. Master’s thesis details

Master’s thesis title in English:

Neural Network Based Eyetracking System

Master’s thesis title in Czech:

Systém sledování očních pohybů založený na neuronových sítích

Guidelines:

1) Review the state of the art methods used in eyetracking.
2) Propose an appropritate system architecture with respect to real-time operation.
3) Implement selected methods and train the system on an dataset of annotated images.
4) Experimentally evaluate speed and accuracy of detection.

Bibliography / sources:

[1] Lin, Horng-Horng, et al. "Pupil localization for ophthalmic diagnosis using anchor ellipse regression." 2019 16th International Conference on Machine Vision Applications (MVA). IEEE (2019)
[2] Vera-Olmos, Francisco Javier, et al. "DeepEye: Deep convolutional network for pupil detection in real environments." Integrated Computer-Aided Engineering 26.1 (2019)
[3] Fuhl, Wolfgang, et al. "BORE: Boosted-oriented edge optimization for robust, real time remote pupil center detection." Proceedings of the 2018 ACM Symposium on Eye Tracking Research & Applications (2018)
[4] Prasad, Dilip K., and Maylor KH Leung. "Clustering of ellipses based on their distinctiveness: An aid to ellipse detection algorithms." 2010 3rd International Conference on Computer Science and Information Technology. Vol. 8. IEEE (2010)
[5] Eivazi, Shaharam, et al. "Improving real-time cnn-based pupil detection through domain-specific data augmentation." Proceedings of the 11th ACM Symposium on Eye Tracking Research & Applications (2019)

Name and workplace of master’s thesis supervisor:

Ing. Jaromír Doležal, Ph.D. CIIRC ČVUT v Praze

Name and workplace of second master’s thesis supervisor or consultant:

Date of master’s thesis assignment: **10.12.2021** Deadline for master's thesis submission: **20.05.2022**

Assignment valid until: **30.09.2023**

Ing. Jaromír Doležal, Ph.D.	prof. Ing. Tomáš Svoboda, Ph.D.	prof. Mgr. Petr Páta, Ph.D.
Supervisor’s signature	Head of department’s signature	Dean’s signature

III. Assignment receipt

The student acknowledges that the master’s thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master’s thesis, the author must state the names of consultants and include a list of references.

.

Date of assignment receipt — Student’s signature

Acknowledgements

First of all, I wish to express my sincere thanks to my supervisor Ing. Jaromír Doležal, Ph.D. for his enthusiasm for the project, for his support, encouragement and patience. Further, I would like to thank the faculty, for providing me with all the necessary facilities for the research. To conclude, I cannot forget to thank my family and close friends for all the unconditional support throughout the study.

Declaration

Abstract

The goal of this thesis is to create a framework for training eye-tracking models. Before the creation of the framework, a comprehensive analysis of methods used for this task was undergone. The work solves a part of the problem using neural networks designed for regression and the other part of the problem using common image processing techniques. Several neural networks were considered, including networks designed for this task but also networks designed for a general regression or classification task. The created framework integrates novel libraries, neural network architectures and methods. The framework is subsequently tested on the dataset of real subjects. Within the testing, the work compares several integrated neural network architectures in terms of their accuracy and speed. Finally, the chosen models are exported to a standard format and inserted in a simple video pipeline that runs in real-time.

Keywords: eye-tracking, pupil detection, neural networks, image processing

Abstrakt

Cieľom tejto práce je vytvoriť framework na trénovanie modelov pre sledovanie očných pohybov. Pred vytvorením samotného frameworku prebehla rozsiahla analýza metód používaných na túto úlohu. Práca rieši časť úlohy pomocou neurónových sietí určených na regresiu a druhú časť pomocou známych metód na spracovanie obrazu. V rámci regresie, boli vzaté do úvahy siete, ktoré sú priamo určené na túto úlohu, ale aj všeobecné siete určené pre regresiu a klasifikáciu. Vytvorený framework integruje novodobé knižnice, architektúry a metódy neurónových sietí. Framework je následne testovaný na datasete reálnych subjektov. V rámci testovania práca porovnáva rôzne architektúry neurónových sietí, ktoré boli vo frameworku integrované, na základe ich presnosti a rýchlosti. Na záver sú vybrané modely exportované do štandardného formátu a zasadené do jednoduchého systému pre spracovanie obrázkov, ktorý beží v reálnom čase.

Klíčová slova: sledování očních pohybů, detekce pupily, neuronové sítě, spracování obrazu

Překlad názvu: Systém sledování očních pohybů založený na neuronových sítích

Contents

Chapter 1

Introduction

Eye-tracking has found its application in a great variety of fields. In medicine, it is an essential tool to assist surgeries, such as laser eye surgery [MEK+01]. It also serves to identify abnormalities in vision and regions of interest of patients with mental diseases such as schizophrenia, bipolar disorder, Alzheimer's disease, and others [HK18]. In psychology, the main fields of application are unquestionably "visual search, reading, natural tasks, scene viewing and other information processing" [MF12]. In marketing, it provides valuable data on eye movements and regions of interest to select and refine products [WP08]. In the virtual reality and gaming industry, eye tracking helps in rendering complex scenes at high frame rates by defining the field of view of a user [CKK19]. It is without a question that one would find many other applications of eye-tracking, especially in recent years.

The task of eye-tracking is to estimate where one is looking, i.e. the gaze vector. Advancements in both software and hardware in recent years allow for non-invasive, real-time and affordable solutions to this task. Thanks to their accessibility, eye-tracking systems gained a great amount of popularity in both the academic and commercial sectors in recent times. Many affordable eye-tracking systems are based on video oculography [KKK+16] [KSM+19a] [WRvA+19]. The gaze vector can be estimated either directly from video frames (images) or indirectly by extracting features from the image and calculating the gaze vector using a geometrical model of the eye. The core of such an indirect feature-based model is an image processing pipeline that takes an image as input and outputs the features required for gaze vector estimation.

This thesis has two main goals: to create a framework for an eye-tracking image processing pipeline that integrates state of the art libraries and to test it on a real-world dataset provided by the supervisor of this thesis by comparing integrated neural network architectures. The final models should work in real-time in an optimized image processing pipeline. The thesis is structured as follows. In Chapter 2, video-oculography eye-tracking methods are introduced. After that, some of the state of the art feature extraction models are depicted. In Chapter 3, the aim of the thesis is described in more detail. Chapter 4 describes the framework built for this work, including integrated libraries, neural network architectures and the process of training them. Models that were chosen to be employed in the final pipeline are explained afterwards. The chapter ends with a description of how these models were evaluated. Consequently, the results are presented in Chapter 5. The results are discussed afterwards in Chapter 6, and suggestions for future improvements are provided. Finally, in Chapter 7, the achieved results of the whole work are summarized.

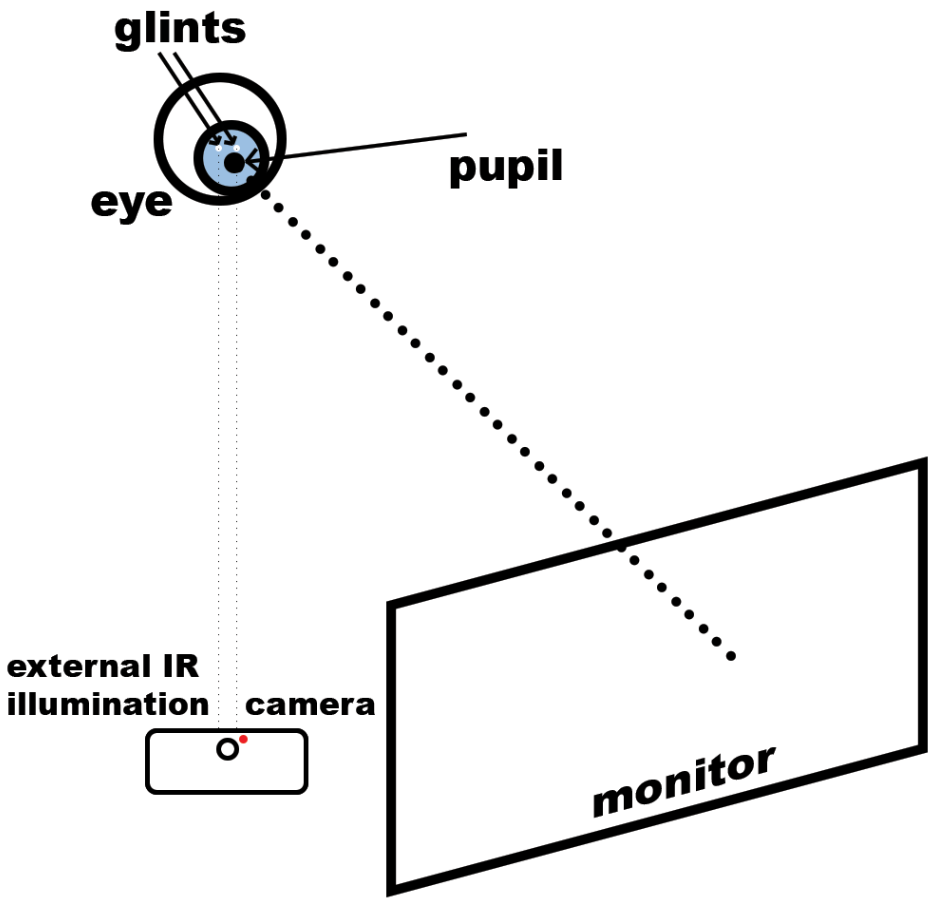

Figure 1.1: **(a) Video oculography eye-tracking system.** The user is looking at the monitor while the camera is sensing his eye movements. Optionally, there's an external infrared illumination reflected from the eye.

Chapter 2

State of the art

Increased interest and demand for affordable eye-tracking systems induced a great wave of research into video oculography models. The next section introduces the video oculography model in great detail. Such models can be classified into two main types [HJ09] [LCM19], namely appearance-based and feature-based methods. As this thesis focuses on feature-based methods due to their better accuracy, appearance-based methods are only outlined. After introducing both methods the NvGaze [KSM$^+$19a] neural network, which was implemented in the framework of this work and chosen for the final models, is described comprehensively.

2.1 Video oculography

Video oculography is a non-invasive method for measuring eye movements. Fundamental parts of an eye-tracking system based on video oculography are a screen and a video camera, optionally accompanied by external infrared illumination (see Figure 1.1). These methods are classified into two main categories [HJ09] [LCM19]: appearance-based methods [TKA02] [SK10] [KKK$^+$16] [KSM$^+$19a] [WRvA$^+$19] and feature-based (also called model-based) methods [LWP05] [YAR$^+$19] [HEM$^+$20]. In general, both methods work as follows: an image is fed to the model, processed through some pipeline that outputs a gaze vector. The following sections cover specifics of the gaze vector retrieval and its representation.

2.1.1 Appearance-based methods

Appearance-based methods [TKA02] [SK10] [KKK+16] [KSM+19a] [WRvA+19] only use a camera to capture an image of eyes and rely solely on the appearance of the eye [HJ09]. The model learns a mapping function from an image directly to a gaze vector. The gaze vector is usually represented as a point on the screen or the rotation angles of the eye regarding the head position [LCM19]. In other words, these models are trained end-to-end and unlike feature-based methods, they do not need any geometrical eye model (see Figure 2.1).

These methods work great under bad light conditions and with low-resolution cameras, where higher robustness is traded for lower accuracy [LCM19]. That is why they found their application in mobile devices and other inexpensive setups [KKK+16]. Best results were achieved using convolutional neural networks [LB+95]. To achieve reasonable accuracy they require a large dataset, which can be either real-world [KKK+16] [KSM+19a] [ZPB+20] or synthetic [KSM+19a]. Convolutional neural network models are capable of achieving real-time performance [KSM+19a] [WRvA+19].

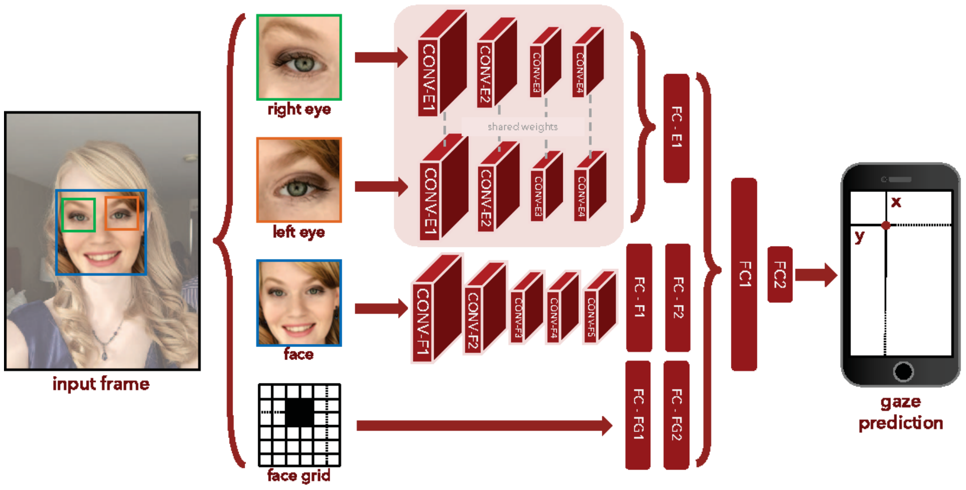

Figure 2.1: Appearance-based methods general principle (from Krafka et al. [KKK+16]) Schema of neural network GazeCaputre [KKK+16] demonstrating end-to-end training of appearance based methods.

2.1.2 Feature-based methods

Feature-based methods consist of two parts: a feature extraction model and a geometrical model. A feature extraction model is trained to extract local features such as contours, eye corners, and eye reflections of the external light sources (further referred to as glints) [LCM19] (see Figure 2.2). Features are then passed to a geometrical model that outputs a gaze vector. The gaze vector is usually represented as a 2D point on the screen (further referred to as the point of gaze) as in the case of appearance-based methods or a 3D direction vector from which a point of gaze can be calculated. This depends on the geometrical model. Here are the main characteristics of both approaches.

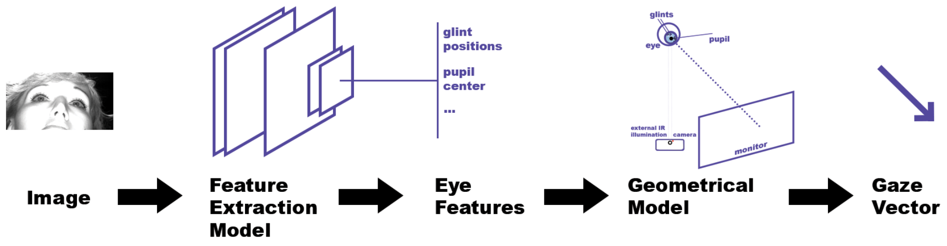

Figure 2.2: Feature-based methods general principle. Eye features such as pupil, cornea and corneal reflections (glints) are extracted from the image. These features are passed to a geometrical model that estimates the gaze vector.

3D approach

The 3D model-based approach yields more robust results but requires a more complex setup in terms of knowing eye parameters that cannot be directly measured and camera parameters [LCM19]. Some of the eye-trackers that use this approach are DeepVOG [YAR+19], RemoteEye [HEM+20] and MagicEyes [WRvA+20]. The 3D approach has two parts. First, locate eye features, i.e. pupil centres and glints. Second, estimate the gaze vector. The first part of this approach is covered in Section 2.2. As for the second part, a geometrical 3D model was developed by Guestrin et al. [GE06]. This model requires at least one camera and at least two external light sources when the head is not stationary and also a subject-specific calibration procedure. More advanced models by Swirski et al. [SD13] and Dierkes et al. [DKB18] overcome these requirements. In addition, all of the methods require that light sources and monitor positions are known when the system is not head-mounted to estimate camera translation and rotation matrices.

As the method by Geusterin et al. [GE06] introduces parameters of the eye that are essential for gaze vector reconstruction, it is described in this section in great detail. After that, the other above mentioned methods are outlined. The geometrical model developed by Guestrin et al. [GE06] for one camera and several external light sources can be derived in the following way (see Eq. 2.1 to 2.14 and Figure 2.3). The derivation is reprinted from [GE06].

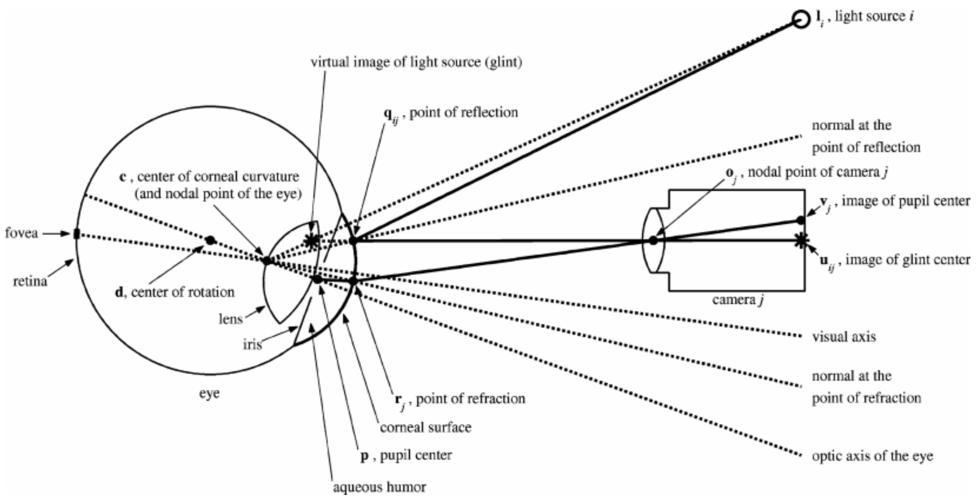
$\mathbf{l}_i$, light source i
virtual image of light source (glint)
$\mathbf{q}_{ij}$, point of reflection
normal at the point of reflection
$\mathbf{o}_j$, nodal point of camera j
$\mathbf{v}_j$, image of pupil center
$\mathbf{u}_{ij}$, image of glint center
camera j
$\mathbf{c}$, center of corneal curvature (and nodal point of the eye)
fovea
retina
$\mathbf{d}$, center of rotation
lens
iris
eye
visual axis
normal at the point of refraction
$\mathbf{r}_j$, point of refraction
corneal surface
$\mathbf{p}$, pupil center
aqueous humor
optic axis of the eye

Figure 2.3: 3D geometric eye model (from Guestrin et al. [GE06]).

Assuming point light sources and pinhole cameras a ray that comes from the light source i, $\mathbf{l}_i$, and reflects at the point $\mathbf{q}_{ij}$ on the corneal surface, modeled as a convex spherical mirror of radius R, such that the reflected ray passes through the camera centre j, $\mathbf{o}_j$, and intersects the camera image plane at the point $\mathbf{u}_{ij}$, following equations can be formulated.

$$\mathbf{q}_{ij} = \mathbf{o}_j - k_{q,ij}(\mathbf{o}_j - \mathbf{u}_{ij}) \text{ for some } k_{q,ij} \tag{2.1}$$
$$||\mathbf{q}_{ij} - \mathbf{c}|| = R \tag{2.2}$$

The law of reflection indicates that points $\mathbf{l}_i$, $\mathbf{q}_{ij}$, $\mathbf{c}$ and $\mathbf{o}_j$ are co-planar and angles $\angle \mathbf{l}_i \mathbf{q}_{ij} \mathbf{c}$ and $\angle \mathbf{o}_j \mathbf{q}_{ij} \mathbf{c}$ are equal:

$$(\mathbf{l}_i - \mathbf{o}_j) \times (\mathbf{q}_{ij} - \mathbf{o}_j) \bullet (\mathbf{c} - \mathbf{o}_j) = 0 \tag{2.3}$$
$$(\mathbf{l}_i - \mathbf{q}_{ij}) \bullet (\mathbf{q}_{ij} - \mathbf{c}) \cdot ||\mathbf{o}_j - \mathbf{q}_{ij}|| = (\mathbf{o}_j - \mathbf{q}_{ij}) \bullet (\mathbf{q}_{ij} - \mathbf{c}) \cdot ||\mathbf{l}_i - \mathbf{q}_{ij}|| \tag{2.4}$$

Similarly, considering a ray that comes from pupil centre $\mathbf{p}$, refracts at the point $\mathbf{r}_j$ on the corneal surface, passes through the camera centre j, $\mathbf{o}_j$, and intersects the camera image plane at the point $\mathbf{v}_j$, following equations can be obtained.

$$\mathbf{r}_j = \mathbf{o}_j - k_{r,j}(\mathbf{o}_j - \mathbf{v}_j) \text{ for some } k_{r,j} \tag{2.5}$$
$$||\mathbf{r}_j - \mathbf{c}|| = R \tag{2.6}$$

The law of refraction indicates that points $\mathbf{p}$, $\mathbf{r}_j$, $\mathbf{c}$ and $\mathbf{o}_j$ are co-planar and from Snell's law (i.e. $n_1 \sin\theta_1 = n_2 \sin\theta_2$) following equations can be formulated.

$$(\mathbf{r}_j - \mathbf{o}_j) \times (\mathbf{c} - \mathbf{o}_j) \bullet (\mathbf{p} - \mathbf{o}_j) = 0 \tag{2.7}$$

$$\begin{split} n_1 \cdot ||(\mathbf{r}_j - \mathbf{c}) \times (\mathbf{p} - \mathbf{r}_j)|| \cdot ||\mathbf{o}_j - \mathbf{r}_j|| = \\ n_2 \cdot ||(\mathbf{r}_j - \mathbf{c}) \times (\mathbf{o}_j - \mathbf{r}_j)|| \cdot ||\mathbf{p} - \mathbf{r}_j|| \end{split} \tag{2.8}$$

where n_2 is the index of refraction of air ($\approx$1).

Finally, considering the distance K between the pupil centre and the centre of corneal curvature leads to

$$||\mathbf{p} - \mathbf{c}|| = K \tag{2.9}$$

If eye parameters (R, K and n_1) are obtained through calibration, substituting Eq. 2.1 into Eq. 2.3 yields the following system of equations.

$$(\mathbf{l}_i - \mathbf{o}_j) \times (\mathbf{u}_{ij} - \mathbf{o}_j) \bullet (\mathbf{c} - \mathbf{o}_j) = 0 \tag{2.10}$$

Assuming single camera is used (i.e. j = 1), the system can be rewritten in matrix from as

$$\underbrace{\begin{bmatrix} [(\mathbf{l}_1 - \mathbf{o}) \times (\mathbf{u}_1 - \mathbf{o})]^T \\ [(\mathbf{l}_2 - \mathbf{o}) \times (\mathbf{u}_2 - \mathbf{o})]^T \\ \vdots \\ [(\mathbf{l}_N - \mathbf{o}) \times (\mathbf{u}_N - \mathbf{o})]^T \end{bmatrix}}_{\text{M}} (\mathbf{c} - \mathbf{o}) = 0 \tag{2.11}$$

From the co-planar constraint (see Eq. 2.10) it follows that matrix M has at most rank 2 and the solution has the following form.

$$\mathbf{c} - \mathbf{o} = k_{c,b}\mathbf{b}_{\text{norm}} \tag{2.12}$$

When $N = 2$ the system can be solved in the following manner.

$$\mathbf{b}_{\text{norm}} = \frac{\mathbf{b}}{||b||} \tag{2.13}$$

$$\mathbf{b} = [(\mathbf{l}_1 - \mathbf{o}) \times (\mathbf{u}_1 - \mathbf{o})] \times [(\mathbf{l}_2 - \mathbf{o}) \times (\mathbf{u}_2 - \mathbf{o})] \tag{2.14}$$

Which is together with calibration parameters sufficient to reconstruct the optic axis $\mathbf{p} - \mathbf{c}$ of the eye.

The downside of this model is that eye parameters (R, K and n_1) must be estimated through several points calibration procedure. The model by Swirski et al. [SD13] solves this problem and requires only a sequence of images from a single camera to estimate necessary parameters, removing also the necessity of external illumination. The core operation of this model is "unprojection" of pupil ellipses from each image into a circle in 3D space, considering full perspective projection. For the full explanation and optimization of the method, see [SD13]. This method was further improved by Dierkes et al. [DKB18] using a more realistic eye model [LG68].

For all of these methods, if the point of gaze is requested, the visual axis needs to be calculated from the optic axis. For doing this, subject-specific parameters have to be estimated, typically from the calibration procedure, in which the subject is required to fixate on a single point [GE06] or these parameters can be approximated by anthropomorphic averages [SD13]. After that, visual axes of both eyes need to be transformed from the image coordinate system to the world coordinate system. To do so, intrinsic and extrinsic camera parameters must be obtained, typically through a camera calibration procedure. For a detailed description of the point of gaze retrieval, see [GE06].

2D approach

The 2D approach is much simpler in terms of initial setup, as no additional information about the position of eye-tracking system parts nor camera calibration are required. This approach includes subject-specific eye parameters and camera parameters directly into a mapping function from 2D feature space like Pupil-Centre-Corneal-Reflections, contours, etc. to the point of gaze [LCM19]. A mapping function can be determined generally, so that no user specific calibration is required, but this approach suffers from lower accuracy caused by head motions. As a result, recent methods employ per subject or even per experiment calibration typically with neural network as a mapping function [WZS16] [HHM+18]. A popular method based on algorithmic approach is Starburst [LWP05].

2.2 Eye feature extraction models

As mentioned earlier, feature-based methods are based on finding distinctive features of the eye to employ them in a geometrical data model. Common features for eye localization include pupil, cornea and corneal reflections, also known as glints. Some notable state of the art algorithms are based on an algorithmic approach [SFK18] [FEH+18] [FGS+18]. The other state of the art models are based on neural networks. Currently, two types of neural networks are commonly used for this task, specifically segmentation networks [YAR+19] [CKA+19] [KCB+21] and regression networks [FSK+17] [KSM+19a] [ESK+19].

The biggest advantage of models based on an algorithmic approach is that they run on CPU within a few milliseconds [FGS+18]. The spectrum of methods employed in these models is very broad. Most commonly used are Canny edge detector [Can86], e.g. [KPB14] [FKS+15] [FSKK16] [SFK18], Haar-like features [VJ01], e.g. [ŚBD12], Gabor filters, e.g. [TKC00] [CK07]. These methods usually deal with an ellipse fitting problem that can be solved, for example, with the RANSAC algorithm [FB81] as in [LWP05] [ŚBD12]. According to NvGaze Paper [KSM+19a], which compares most of the methods mentioned above on the PupilNet dataset [FSK+17], Circular binary features algorithm [FGS+18] or CBF was the only method that yielded similar results as the neural network based NvGaze and outperformed it in terms of precision when pixel error threshold for detection was lower (see Figure 2.4). Besides high accuracy, CBF runs at a very short inference time of 0.44 milliseconds to 6.8 milliseconds on a CPU, depending on the resolution and approach, which is lower or very similar in comparison to other models [FGS+18].

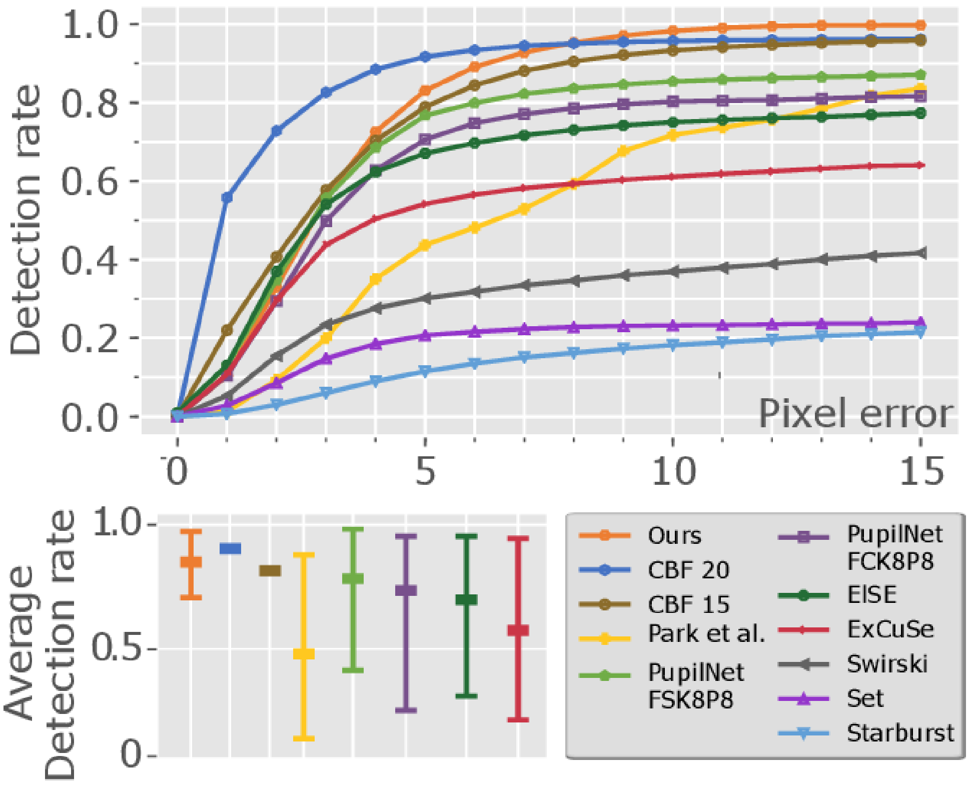

Figure 2.4: Comparison of state of the art feature extractors (from Kim et al. [KSM+19a]). Original description, where "We, Our and Ours" refer to NvGaze model [KSM+19a]: "Average pupil estimation error on PupilNet datasets. Top: We compare the average detection rate of our pupil estimation network against Starburst [LWP05], Set [JHB+15], Swirski et al. [ŚBD12], ExCuSe [FKS+15], ElSe [FSKK16], PupilNet [FSK+17], Park et al. [PZBH18] and CBF [FGS+18]. Bottom: The 5-pixel error is averaged across individual PupilNet datasets [FSK+17] (bold marker) and bounded by best and worst error values for all datasets (upper and lower markers). Our approach reaches highest robustness. Note that for CBF only the average detection rates over all datasets were published, not the detection rate for individual datasets." The metrics is described as follows, "Accuracy of pupil estimation is usually given in form of a probability of estimating the pupil location with a maximum distance of 5 pixels from the ground truth pupil location (”5-pixel error” or "detection rate")" [FSK+17].

On the other hand, the use of neural networks for this task becomes more and more popular lately [GSS+19]. In short, neural networks, as the core of deep learning, process input vector, e.g. an image, through several layers of pre-trained weights and output a prediction vector. The weights are trained through learning procedure called backpropagation [RHW86]. As previously stated, there are two main categories of neural networks. The first one treats the problem as a segmentation problem. Neural networks in this category employ Unet-like [RFB15] architecture that consists of an encoder network and a decoder network with skip connections and pixel map as output [YAR+19] [CKA+19] [KCB+21]. In the output pixel map, each pixel is assigned to a category, i.e. pupil, glint. For an illustration of such a network see Figure 2.5 which shows the architecture of the RitNet [CKA+19] neural network, that won the OpenEDS challenge [GSS+19]. The network can process 301 frames per second on an NVIDIA 1080Ti, which corresponds to a processing time of 3.3 milliseconds [CKA+19].

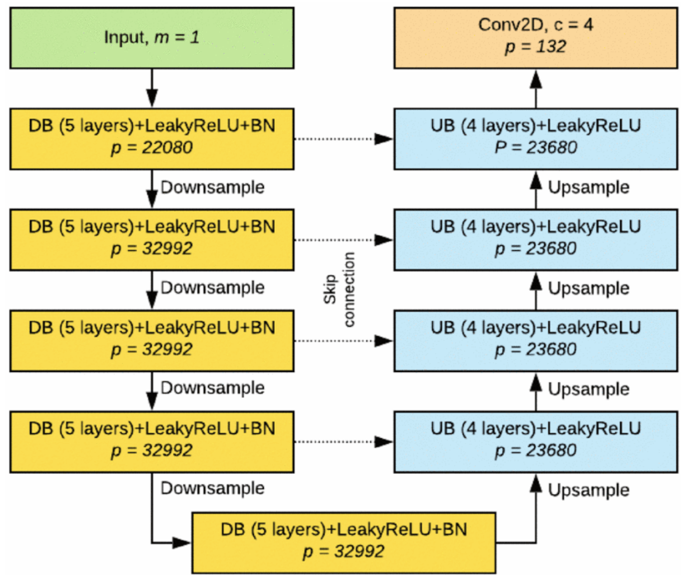
Input, m = 1
DB (5 layers)+LeakyReLU+BN p = 22080
Downsample
DB (5 layers)+LeakyReLU+BN p = 32992
Downsample
DB (5 layers)+LeakyReLU+BN p = 32992
Downsample
DB (5 layers)+LeakyReLU+BN p = 32992
Downsample
DB (5 layers)+LeakyReLU+BN p = 32992
Skip connection
Conv2D, c = 4 p = 132
UB (4 layers)+LeakyReLU P = 23680
Upsample
UB (4 layers)+LeakyReLU p = 23680
Upsample
UB (4 layers)+LeakyReLU p = 23680
Upsample
UB (4 layers)+LeakyReLU p = 23680
Upsample

Figure 2.5: Segmentation neural network for eye feature extraction - RitNet (from Chaudhary et al. [CKA+19]). Original description: "Architecture details of RITnet. DB refers to Down-Block, UB refers to Up-Block, and BN stands for batch normalization. Similarly, m refers to the number of input channels (m = 1 for gray scale image), c refers to number of output labels and p refers to number of model parameters. Dashed lines denote the skip connections from the corresponding Down-Blocks. All of the Blocks output tensors of channel size m=32." [CKA+19]

The second category treats the problem as a regression problem. Networks in this category usually employ only an encoder convolutional neural network [LB+95] with the last layer outputting eye features, i.e. position of the pupil centre, position of glints, etc. [FSK+17] [KSM+19a] [ESK+19]. These networks are often inspired by networks used for image classification or parts of them, e.g. PupilNet [FSK+17] is based on Inception [SLJ+15]. A neural network from this category, that is also used in this work is called NvGaze [KSM+19a] and is described in the next section 2.3.

2.3 Regression neural network for eye feature extraction - NvGaze

The NvGaze neural network for pupil localization was designed to be as simple as possible in order to achieve a short inference time [KSM+19a]. The backbone of the architecture consists of 7 convolutional layers (see Table 2.1) using dropout with the probability of 0.1 and ReLU activation. After the backbone, there is a linear (fully-connected) layer also using dropout (p=0.1) (see Figure 2.6) [KSM+19b].

Layer index	1	2	3	4	5	6	7
Kernel size	9x9	7x7	5x5	5x5	3x3	3x3	3x3
Output channels	24	36	52	80	124	256	512

Table 2.1: NvGaze backbone from Kim et al. [KSM+19b].

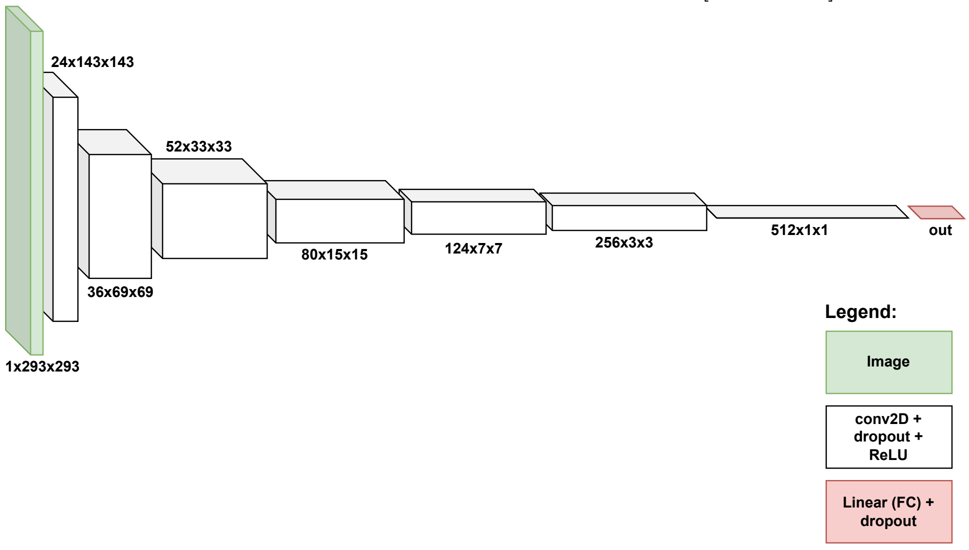

Figure 2.6: NvGaze Architecture. An image is fed to the pipeline of 7 convolutional layers with dropout and ReLU activation followed by a linear (fully-connected) layer using dropout with the probability of 0.1. Each block symbolizes the output with the shape below it of a layer represented by a color.

As the architecture is very simple, the core of the network lies in the data augmentation of a large synthetic and real-world dataset. The data augmentation is described as follows: "First, we transform the image and label using a random affine transformation (translation up to 50 pixels, rotation about origin up to 10°, rescale with a randomized factor within the range [0.9, 1.5]) while making sure that pupil location is within a reasonable range of [0.1, 0.9] of horizontal and vertical image size. Second, we apply pixel-wise intensity noise with a maximum offset of ±10 (probability p=0.5). Third, we apply a global intensity offset of ±40 (p=0.5). Fourth, we apply Gaussian filtering with $\sigma \in [0.6, 1.6]$ and kernel size of 7 (p=0.5). Fifth, using bicubic filtering we apply random shrinking with a scale factor s $\in$ [0.25, 1.0] followed by upscaling again to the full input resolution (p=0.5). Sixth, to simulate environment reflections in the eye we randomly overlay the image with images out of 326 natural photographs from the dataset published in [7] (p=0.25). Before blending, the typically larger overlay image B is converted to grayscale and randomly cropped the eye image resolution. For eye image E and a randomly chosen opacity value $o \in$ [0.1, 0.2] we use a soft blending function $(1-((1-E)*(1-B)))*o+E*(1-o)$ and clip the resulting pixel intensities to a maximum of 255. Seventh, we apply histogram equalization implemented in OpenCV [Bra00]. As a last step, we rescale the image intensities to a range of [-1, 1] and randomly shift the mean about ±0.15 and the min and max intensities about ±0.1. During inference of test images only histogram equalization and normalization to [-1, 1] are applied." [KSM+19b].

The architecture was chosen for the final models for this thesis. Its modifications and parameters are described in Section 4.2.

Chapter 3

Aim of the thesis

As the demand for accurate and fast eye-tracker increased over the past few years, a very broad research is being led in this field. In Chapter 2, several methods for eye-tracking were introduced. The more accurate methods required the extraction of eye features, i.e. a pupil centre or corneal reflections. Consequently, several eye feature extraction models were introduced in Section 2.2 and 2.3. Many of the novel extractors were based on neural networks for segmentation or regression. It is noteworthy that not only neural networks made for eye-tracking but also neural networks for general tasks such as classification, regression and segmentation can be used for this task with slight adjustments.

Considering the fast development in eye-tracking, the aim of this thesis is not simply to implement a solution for eye feature extraction but to create a framework that provides simple integration of novel neural networks. The framework is tested on a dataset provided by the supervisor of this thesis by comparing some integrated neural network architectures and choosing the final models that can be integrated into an image processing pipeline.

With this abstraction, the parts of the framework like a dataset or a model can be changed, but the methods that are common for these parts, e.g. extracting smaller patches of both eyes, measuring errors and plotting results do not need to be changed. The framework should be based on the state of the art libraries like PyTorch [PGM$^+$19] to easily integrate new methods for training neural networks and employ modern pre-trained neural networks. Trained models should be exported into a standard format such as ONNX [dev21] to be easily integrated into an inference pipeline.

Chapter 4

Methods

This chapter begins with the description of the framework for training neural networks used for eye feature extraction, and examples of how the framework was used for the provided dataset. After that, the parts of the chosen model are described, including the network for pupil detection and the algorithm for glint detection. The chapter ends with the description of methods for measuring detection accuracy and inference time.

4.1 Framework

As the research in machine learning advances very fast, it is important to implement neural networks using up-to-date libraries. The advantage of this approach is that these libraries usually integrate advancements in deep learning research quickly and often more efficiently. Another advantage is that if a developer wants to try a new feature or a model and doesn't know if it brings any improvement, it can be already integrated with one of the employed libraries. As a base for this thesis, a library called fastai [HG20] was chosen. Based on PyTorch [PGM+19], NumPy [HMvdW+20], PIL [CC21] and other libraries, fastai provides several functionalities from a high-level API such as the one-cycle policy [Smi18] to low-level API to modify the behaviour of the training pipeline. In addition, one can use the PyTorch methods directly [HG20]. Besides fastai, the framework integrates the timm library [Wig19] to easily access common neural networks for image classification and segmentation that were pre-trained on the Imagenette dataset [How]. Furthermore, it integrates the vit-pytroch [Wan21] library to simply access the vision transformer [DBK+20] model and its modifications. Notably, integration of other model libraries is very straightforward as long as the models are written in PyTorch. The framework also integrates the albumentations [ABK18] library based on OpenCV [Bra00] for complex data augmentation. For tracking the learning and results, the wandb library is used. The library appeared to be very helpful in visualizing the learning progress and comparing the results of different models. To test the inference of the model, a simple OpenCV [Bra00] pipeline is used that accepts models in ONNX format [dev21] and allows the user to add simple python algorithms, e.g. for glint detection.

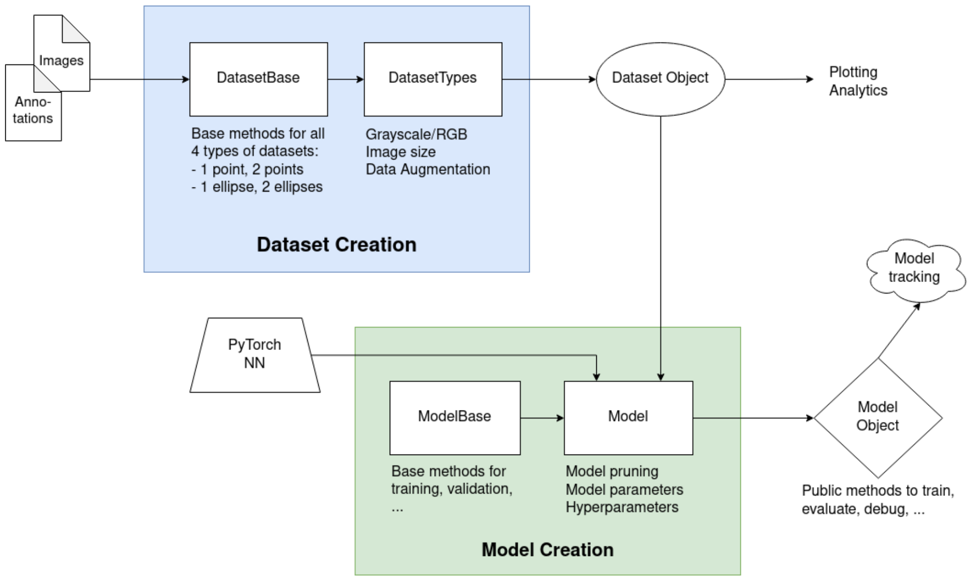

Figure 4.1: Training Pipeline. In **Dataset Creation**, several datasets are created for various tasks from data files, e.g. finding pupil centres, ellipses fitting for one or two eyes and possibly others. Several model-specific parameters can be set, such as the number of image channels, image size or data augmentation. Finally, a Dataset object containing DataBlock from the fastai library [HG20] is created and forwarded into the model. **Model Creation** accepts a neural network in PyTorch [PGM+19] format and provides methods, some of them based on fastai library [HG20], to train the model, find the best learning rate, etc. It outputs the model object, which is used for final interaction in the training, evaluation and analysis. The training and evaluation of the model are tracked using wandb [Bie20] cloud service.

In general, the training pipeline can be described as follows (see Figure 4.1). The pipeline starts with a dataset object that is described in Section 4.1.1 in more detail. The core method of this object serves the purpose of creating a fastai [HG20] DataBlock object. This provides a great level of abstraction since the fastai DataBlock is in a standard form and fastai takes care of creating a DataLoader object for training, validation and testing. Furthermore, common methods such as calculating the mean and variance of the dataset can be written regardless of the actual file representation of the dataset. Most importantly, the DataLoader is passed to a neural network model. The Model object is described exhaustively in Section 4.1.2. Shortly, a fastai model, which can be created from PyTorch [PGM+19] model with a simple wrapper, expects a DataLoader object, which is standardized. In this manner, the model is developed separately from the dataset. For this object, the user can use built-in fastai methods, such as schedulers for model parameters, callbacks to track the progress of learning and weights, etc. [HG20]. Moreover, custom user functions such as plotting results in a standard form or tracking custom errors may be used. Details about the training and hyperparameters are provided in Section 4.1.3. Lastly, the final ONNX [dev21] model is exported for inference (see Section 4.1.4), along with results from the test dataset (see 5.1) and benchmark results (see 5.2). These are described in Section 4.3.

4.1.1 Dataset creation

The dataset provided by the supervisor of this thesis contains 5823 grayscale images of the head with the resolution of 2048x1088 pixels of 634 subjects of adults and children from indoor rooms from various places with various lighting conditions and external IR illumination (see Figure 4.2). In comparison, Pupilnet 2.0 dataset [FSK+17] contains 135k images of the eye with resolution 384×288 and the NvGaze dataset [KSM+19a] contains 2M synthetically generated (3D modelled) images at 1280×960 and 2.5M real images at 640×480. For simplicity, all pupil borders are annotated with circles. However, for complexity, future work and coherence with public datasets, they are treated as ellipses. The dataset contains images of open eyes only, i.e. the pupil centre is visible.

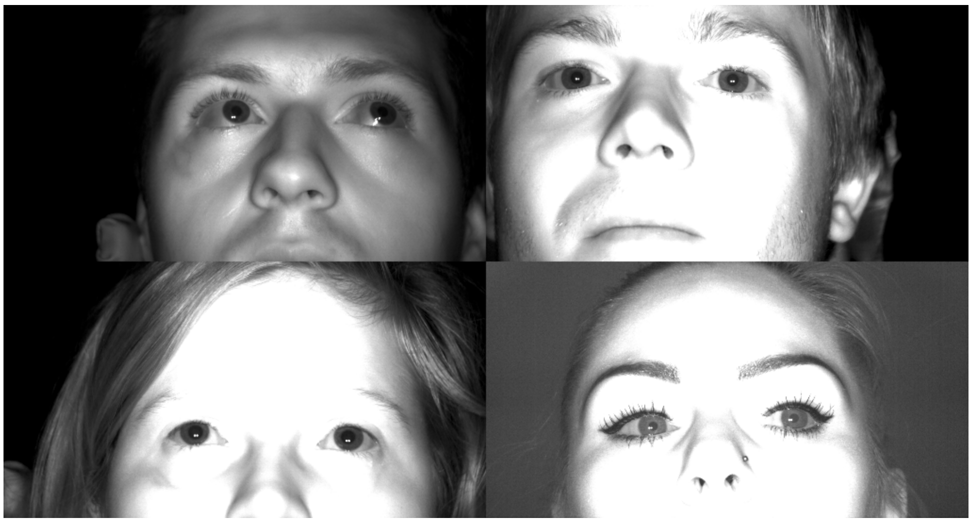

Figure 4.2: Examples of the dataset provided by the supervisor of the thesis.

To achieve the best accuracy considering the data, the problem was split into 3 tasks:

1. Estimate pupil centre position to extract patches around the eyes from full resolution image [neural network]
2. Given an eye patch, fit an ellipse to the pupil [neural network]
3. Given an eye patch, locate corneal reflections (glints) [algorithmic]

For the first and the second task, several neural networks from Section 4.1.2 were considered. These will be further explained in this section. As the position of the glints is not annotated in the dataset, common image processing techniques are used to locate the glints. The pipeline for glint localization is explained in Section 4.2.3.

For the first task, the original dataset was used. The images were split into the train folder, which is used for updating the weights during the training of a neural network, the validation folder, which is used to choose the best model during the training of the network, and the test folder, which is used to compare different models. The splitting was done in a way that all images of a single subject are in only one of the three folders (see Figure 4.3).

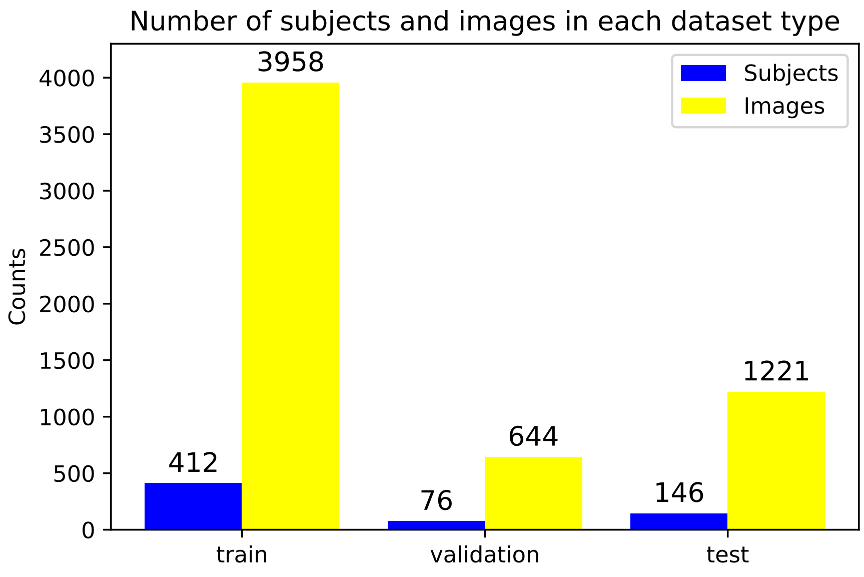

Figure 4.3: Counts of images and subjects in dataset folders.

As for the second task, patches around the pupil centres were extracted (see Figure 4.4). These patches can be rectangles or squares. In this work, square patches were extracted with the size according to the requirements of neural networks. Considering a temporary patch with the centre in the pupil centre, a random position within this patch was chosen as the final patch centre, ensuring that the new patch is not outside the image border and the pupil and glints are visible. Put differently, the patches around the pupils were extracted, so the position of the pupil in the patch is random, but the pupil is still completely visible.

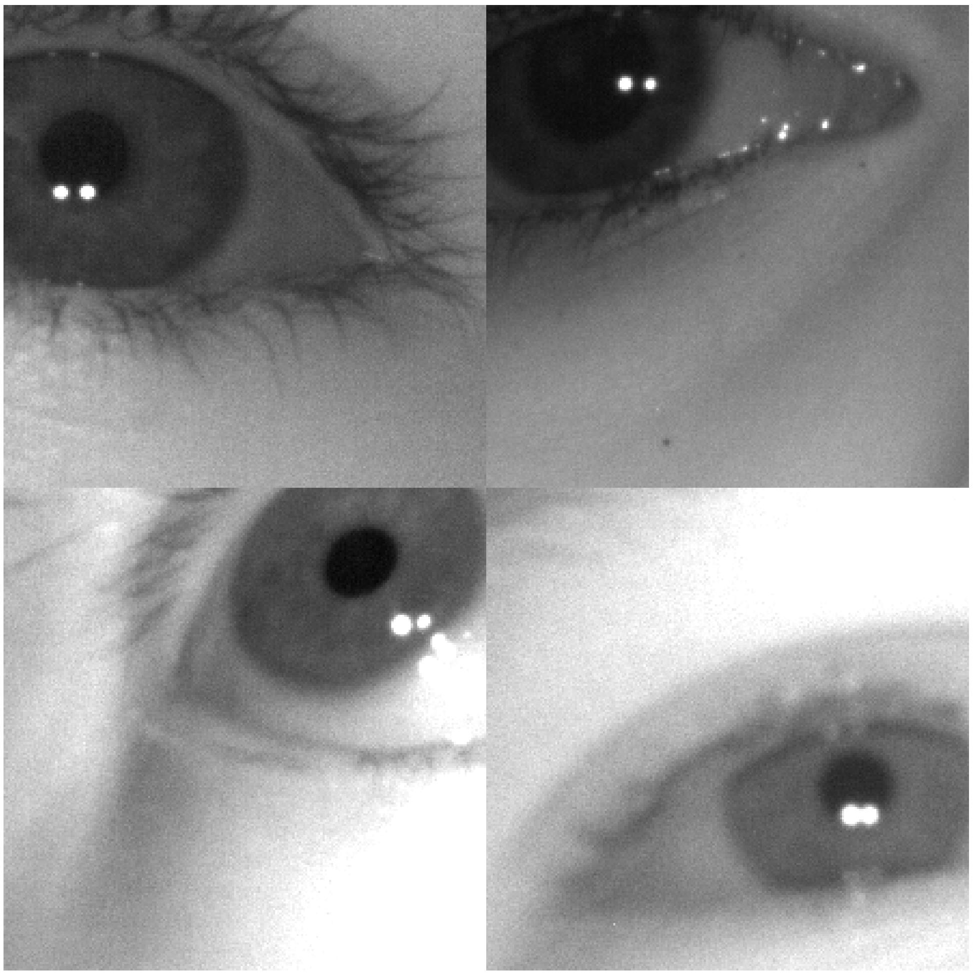

Figure 4.4: Examples of the dataset of randomly extracted pupils. The dataset was used to train the algorithm for fitting the ellipse around the pupil.

At this point, the files are prepared. The goal is to create a unified dataset object (see Figure 4.1) from the actual dataset representation for each part of the model. For the pupil patch extraction part, the label is created only from the pupil centres - 2 points. As for the ellipse fitting part, there are several approaches. The image patches of the left and right eye can be concatenated, resulting in a rectangular image with 2 eyes. Another approach would be to stack them in a batch of two and detect the ellipse on the each eye separately.

Also, instead of fitting ellipses directly, there may be a neural network that only finds the pupil centres, and another algorithm will be used for ellipse fitting. For this reason, the user can specify whether he wants to treat the label of the dataset as one point, 2 points, one ellipse or two ellipses. Methods for dataset loading, augmentation, plotting, etc., were thus written for all 4 types of dataset representations.

There are also several ways of manipulating the images. They can be loaded as grayscale images as they are, or the channel can be copied 3 times to simulate a 3-channel image. This feature is important for neural networks pre-trained on the Imagenette dataset [How] as they expect the input to have 3 channels. There is also an option to take the mean along the 3 channels and train the model on grayscale dataset. The user can decide the image size, which is done by maintaining the aspect ratio. If the network requires a square patch, as in the case of vision transformers [Wan21], the image is 'squished' along the bigger dimension. Common interpolation techniques are available, although only bilinear interpolation is used for the purpose of this thesis. The user can also provide values for dataset normalisation, i.e. the mean and variance of the dataset. Lastly, the user can specify which data augmentations will be employed. The augmentation process is described in the following Section 4.1.1. As everything is set, the fastai DataBlock [HG20] is created and passed to the model. Moreover, the DataBlock, as it is configured for the current scenario, can be used in functions such as plotting or extracting patches of the eye.

Data augmentation

As it was outlined in Section 2.3, data augmentation plays an important role in neural network training. It is the technique of extending the dataset by employing common image processing algorithms to generate new images from the original image. These techniques can be chained with a probability that each of them will be applied. The fastai [HG20] and PyTorch [PGM$^+$19] libraries provide such data augmentations natively, but for the main library, Albumentations [ABK18] was chosen. The reason for that is that this library provides support for affine transformations of the image, including keypoints, and it could be easily modified to transform ellipses. Several augmentation techniques were considered and tested. Following data augmentations were chosen for the final model comparison with the following parameters and probabilities (see List 4.1.1 and Figure 4.5).

- Shift, Scale, Rotate - (image and label), where the shift limit is 0.2 of the biggest dimension, scale limit is 0.9x - 1.1x of the original image size, rotation limit is 10, p=0.75
- Gaussian Blur, with kernel size $\in \{5, 6, 7\}$ and variance $\in [0.6, 1.6]$, p=0.5
- Gaussian Noise, with mean 0 and variance 9, p=0.5
- Random Brightness and Contrast, with a brightness and contrast of limit 0.3 (library-specific)
- Random Shadow (only with 3-channel images)

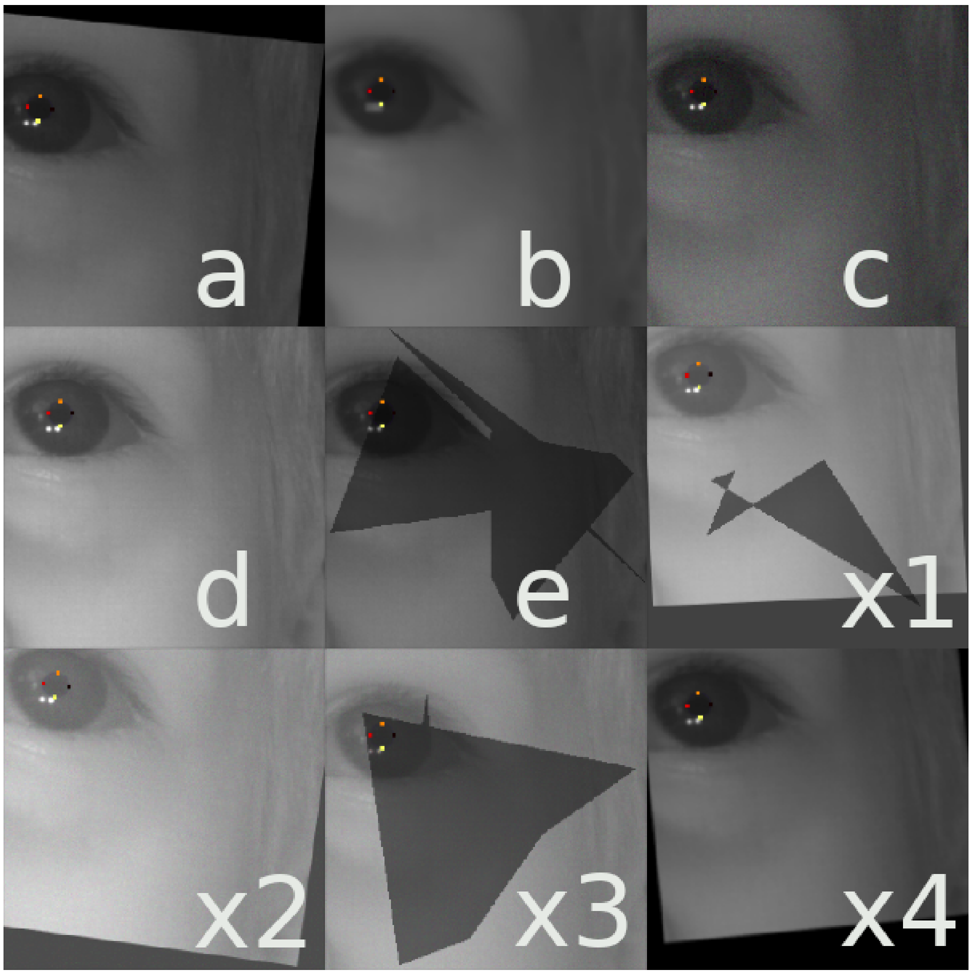

Figure 4.5: Examples of data augmentation. The four colourful points are points of the intersection of the major and minor axes with the ellipse surface. **a)** Shift, Scale, Rotate **b)** Gaussian Blur **c)** Gaussian Noise **d)** Random Brightness and Contrast **e)** Random Shadow **x1-x4)** images with random augmentations generated with correct parameters as in List 4.1.1.

Augmentations are used only during the training part (not during the validation). Shif, Scale, Rotate, as it suggests, shifts, scales, and rotates the image with the label. If the transformation would make the pupil out of the visible part of the image, the transformation is discarded. Gaussian Blur convolves the image with a Gaussian kernel of a randomly chosen size from $\{5, 6, 7\}$ and randomly chosen variance in interval $[0.6, 1.6]$, which is calculated as in Eq. 4.1. Gaussian Noise introduces pixel and channel-independent noise. Random brightness and contrast simulates the varying lighting conditions by adjusting the image histogram. Finally, Random Shadow simulates shadows by overlaying an image with a polygon. It was developed in this [Sax18] library.

$$G(x, y) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{4.1}$$

where σ is variance of the distribution.

4.1.2 Model creation

As can be seen in Figure 4.1, the training pipeline accepts any PyTorch model [PGM+19]. Most of the popular neural networks are already implemented using PyTorch, like vision transformers [DBK+20] [Wan21] and their modifications. Some of them are already pre-trained on the Imagenette dataset [How], and those are available, for example, in timm library [Wig19]. Training a model, that was already trained on another dataset is called transfer learning [TS09]. It is usually done by freezing the first layers of the model as these learn the more general features and updating weights only of the final layers of the model for a few epochs. After that, the layers are unfrozen, and the model trained is trained as a whole. This procedure is also known as fine-tuning the model. The procedure not only improves the training time but can potentially lead to better results [PY09]. In addition, this work implements the NvGaze [KSM+19a] model for pupil centre extraction in PyTorch [PGM+19] as this simple neural network architecture showed promising results with fast inference time. Altogether, 87 models were trained from the above libraries and tracked using the wandb [Bie20] library. From the first 67 models, the most promising were chosen for the final comparison together with the chosen data augmentations (see Section 4.1.1). Bearing in mind how the problem was split (see List 3), here are the networks considered for each task employing a neural network.

For the extraction of the eye patch, which is based on estimating the pupil centre on the whole 2048x1088 image of the head downscaled to the input size of the neural network, the following neural networks were considered (see

Table 4.1). The same dataset was used and extended through augmentation (see Section 4.1.1) differing only in dimensions to match the input size of the model. Hyperparameters were mostly identical (see Section 4.1.3). Results of the comparison are presented in Figure 5.1.

Base model [created | implemented]	Input size [CxWxH px]	parameters	batch size	epochs (full | frozen)
Resnet34 [HZRS16] [HG20]	3x256x136	21,8M	64	300 | 10
Resnet34 [HZRS16] [HG20] (gs)	1x256x136	21,8M	128	300 | 10
NvGaze [KSM+19a] (B)	3x512x512	1,8M	64	600 | x
NvGaze [KSM+19a] (gs, B)	1x512x512	1,8M	64	600 | x
NvGaze [KSM+19a] (gs)	1x293x293	1,8M	128	600 | x
ViT Tiny [DBK+20] [Wan21]	3x224x224	5,5M	128	400 | 10
ViT Small [DBK+20] [Wan21]	3x224x224	22,5M	128	400 | 10
CCT [HWS+21] [Wan21] (gs)	1x256x256	26M	32	600 | x

Table 4.1: Neural networks considered for extracting eye patches. Input size denotes the number of channels and width and height in pixels. The batch size and learning rate were adjusted according to the model and according to the GPU memory limit. Epoch means that the whole training dataset was processed once. The 'x' symbol at frozen epochs denotes that the network was not pre-trained. The number of epochs was set empirically with the goal to keep reasonable training time, i.e. less than one day, obtain comparable results and get close to convergence. Symbols in parenthesis after the name denote: B - the bigger version of the network, from listed; gs - the network accepts grayscale input. All models are written in PyTorch [PGM+19] library.

The second part is the one where the eye patch is fed to the input of the neural network and outputs the ellipses fitted to the pupil. One patch contains a single eye. For this task, the following networks were considered, mainly due to their inference time discussed in Section 5.2. Again the same dataset was used and extended through augmentation, (see Section 4.1.1) differing only in dimensions to match the input size of the model. Hyperparameters were mostly identical (see Section 4.1.3). Results of the comparison are presented in Figure 5.2.

Base model [created | implemented]	Input size [CxWxH px]	#parameters	batch size	# epochs (full | frozen)
Resnet34 [HZRS16] [HG20]	3x256x256	21,8M	128	1000 | 10
NvGaze [KSM+19a] (gs)	1x293x293	1,8M	64	1500 | x
NvGaze FC [KSM+19a] (gs)	1x293x293	1,9M	64	1500 | x
ViT Tiny [DBK+20] [Wan21]	3x224x224	5,5M	128	2000 | 10

Table 4.2: Neural networks considered for pupil ellipse fitting. Input size denotes the number of channels and width and height in pixels. The batch size and learning rate were adjusted according to the model and according to the GPU memory limit. Epoch means that the whole training dataset was processed once. The 'x' symbol at frozen epochs denotes that the network was not pre-trained. The number of epochs was set empirically with the goal to keep reasonable training time, i.e. less than one day, obtain comparable results and get close to convergence. Symbols in parenthesis after the name denote: gs - the network accepts grayscale input. All models are written in PyTorch [PGM+19] library. NvGaze FC is described in Section 4.2.2.

When the PyTorch [PGM+19] neural network module is prepared, the fastai [HG20] is used to prepare the model for a given scenario in the Model block (see Figure 4.1). The dataset object is retrieved and configured for the given scenario. If the model is not directly suitable for the task, unlike the NvGaze [KSM$^+$19a] model, the model needs to be modified. The fastai [HG20] library provides simple methods to do that. At first, the model is pruned. This means that part of the network, usually, the last layers of it, also called the head, is removed. If the model is pre-trained, the weights of the layers that were trained are preserved. After that, the new regression head is created and initialized using e.g. kaiming initialization [HZRS15]. A simple regression head consists of several fully connected layers, with the last layer outputting the target with predicted values. For more details of the architecture of the final model see 4.2. Next, model parameters are set, e.g. the complexity of the model, input size, optimizer, loss function, as well as hyperparameters and their schedulers such as the learning rate, momentum or weight decay.

When everything is set, the final model object is created. In this way, the training pipeline exposes only two objects for further interaction (see Figure 4.1). The dataset object contains methods specific only to the dataset, e.g. extracting a new dataset of random eye patches of a given size, plotting the dataset and calculating statistics of the dataset. As the model is tied to the given dataset object (a model that is used for dataset patch extraction cannot be used with the dataset object used for ellipse fitting), the methods for model object often require both model and dataset. These are finding a suitable learning rate, training the model, fine-tuning it, plotting results, measuring errors, extracting dataset patches around the estimated pupil location or exporting the model. A huge advantage of integrating the fastai library [HG20] was that some of these methods are just simple wrappers around fastai methods and did not need to be rewritten. Contrary, if the methods are not available, they can be written using PyTorch [PGM$^+$19] because the fastai library only extends the PyTorch library.

As the whole training pipeline with its options was introduced and the datasets were explained, the only missing thing is the setting training parameters. Assuming the models are established with their model-specific parameters, (see Tables 4.1 and 4.2) hyperparameters must be set. The procedure in this thesis was to determine some default hyperparameters and choose a proper neural network architecture for the task. When chosen, hyperparameters were adjusted to improve the accuracy on the test dataset. The following section describes functions and hyperparameters used in the comparison of the models.

4.1.3 Training

Considering Figure 4.1, previous sections discussed the dataset and the model creation. The model object with default parameters can be used to plot schedulers and find a proper learning rate so that the model is adjusted and a new model object can be created. This section describes the training procedure, hyperparameters and schedulers.

In essence, a neural network takes an image as an input, processes it through its architecture containing pre-trained weights and outputs a prediction vector. Weights are trained through a learning procedure called backpropagation [RHW86]. Considering one task from the extraction pipeline (see List 3), the common part of backpropagation is a loss function.

Loss function and batch size

The loss function calculates the error between the predicted value from a neural network $\mathbf{x}$ and the target value $\mathbf{y}$.

For the localization of the pupil centre L2 distance loss function was used (see Eq. 4.2).

$$l(\mathbf{x}, \mathbf{y}) = (\mathbf{x} - \mathbf{y})^2 \tag{4.2}$$

where square is taken element-wise ($\mathbf{x}^2$ indicates $\mathbf{x} \odot \mathbf{x}$).

For the ellipse fitting part, the loss function is more complex. The loss function was inspired by the work from Sanchez-Lopez et al. [SLCLV20]. The ellipse can be represented as $\mathbf{x} = [x, y, r_x, r_y, \theta]^T$, where $x, y \in [0, 1]$ denotes the position of the ellipse centre in the image relative to the image size, $r_x, r_y \in [0, 1]; r_x \geq r_y$ denotes the length of the x-major axis and length of the y-minor axis relative to the image size and $\theta \in [-\pi/2, \pi/2]$ denotes the angle of rotation of the ellipse in a counter-clockwise direction. The loss function calculates the L2 distance between the 4 points defined by the intersection of major and minor axes with the ellipse surface (see Eq. 4.3 and 4.4. See also Figure 4.6).

$$f(\mathbf{x}) = \begin{bmatrix} x \\ y \\ x \\ y \\ x \\ y \\ x \\ y \end{bmatrix} + \begin{bmatrix} r_x \\ r_x \\ r_x \\ r_x \\ r_y \\ r_y \\ r_y \\ r_y \end{bmatrix} \odot \begin{bmatrix} \cos\theta \\ \sin\theta \\ -\cos\theta \\ -\sin\theta \\ \sin\theta \\ -\cos\theta \\ -\sin\theta \\ \cos\theta \end{bmatrix} \tag{4.3}$$

$$l(\mathbf{x}, \mathbf{y}) = (f(\mathbf{x}) - f(\mathbf{y}))^2 \tag{4.4}$$

where $\odot$ denotes element-wise multiplication and square is taken element-wise ($\mathbf{x}^2$ indicates $\mathbf{x} \odot \mathbf{x}$).

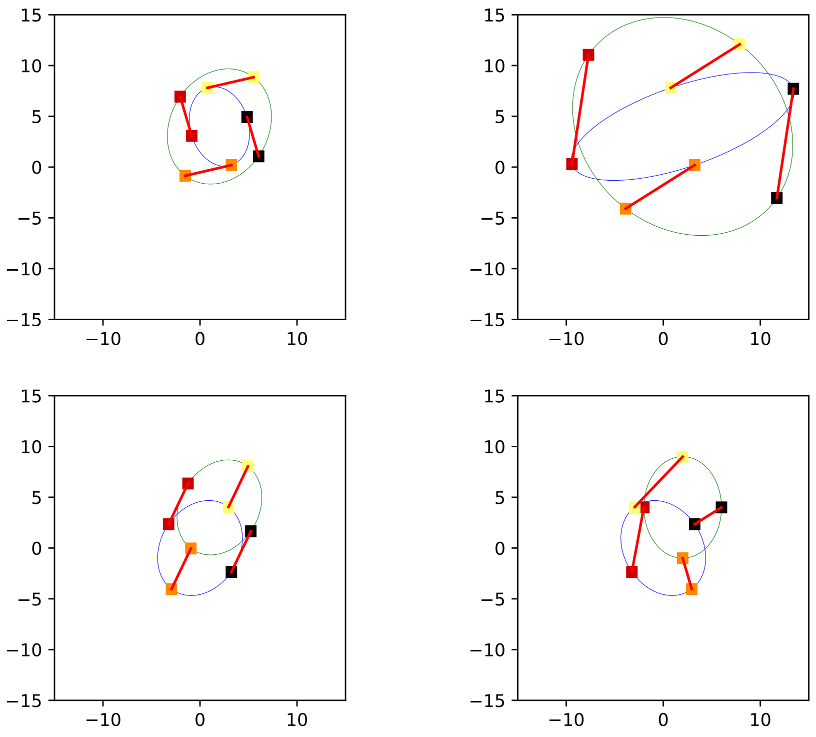

Figure 4.6: Intuition behind the ellipse loss. The four colourful points are points of the intersection of major and minor axes with the ellipse surface. The red line indicates the distance between these two points. Note that in the actual loss, the distances are squared.

In both cases, items (images with labels) in the dataset are processed in batches of some batch size, which is a hyperparameter. As the networks were compared, the biggest possible batch size, considering GPU limit, from the set $\{32, 64, 128\}$ was chosen. The advantage of a bigger batch size is shorter training time. A disadvantage is that bigger batch size tends to yield worse results regarding the test dataset [KMN+16], but this can be partly compensated by adjusting the learning rate [GDG+17]. It is also important to note that the final batch size is adjusted when the final model architecture is chosen.

Knowing how to calculate the loss of a single vector, the final loss function of the batch can be defined by the sum of single items with some reduction. Mean reduction was used for both experiments (see Eq. 4.5)

$$L(\mathbf{b}_x, \mathbf{b}_y) = \frac{1}{nm} \sum_{i=1}^{n} \sum_{i=1}^{m} l(\mathbf{b}_x(i), \mathbf{b}_y(i)) \tag{4.5}$$

where n is the batch size and m is the number of elements in a vector $l(\mathbf{b}_x(i), \mathbf{b}_y(i))$.

Optimizer

For all experiments, AdamW optimizer [LH17] (see Eq. 4.6-4.11) was chosen as the base optimizer. It is basically an Adam optimizer [KB14] which addresses the problem of weight decay and L2 regularization implementation in popular neural network training libraries, so the regularization is performed correctly.

$$g_t = \nabla_\theta f_t(\theta_{t-1}) + \lambda\theta_{t-1} \tag{4.6}$$
$$m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t \tag{4.7}$$
$$v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2 \tag{4.8}$$
$$\hat{m}_t = \frac{m_t}{(1 - \beta_1^t)} \tag{4.9}$$
$$\hat{v}_t = \frac{v_t}{(1 - \beta_2^t)} \tag{4.10}$$
$$\theta_t = \theta_{t-1} - \alpha \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} - \lambda\theta_{t-1} \tag{4.11}$$

where f_t is an objective function with parameters θ, g_t^2 indicates square $g_t \odot g_t$, and all operations on vectors are element-wise. α is the learning rate hyperparameter, β_1, β_2 are momentum hyperparameters, and ϵ is another hyperparameter [KB14]. The hyperparameter λ is weight decay from [LH17].

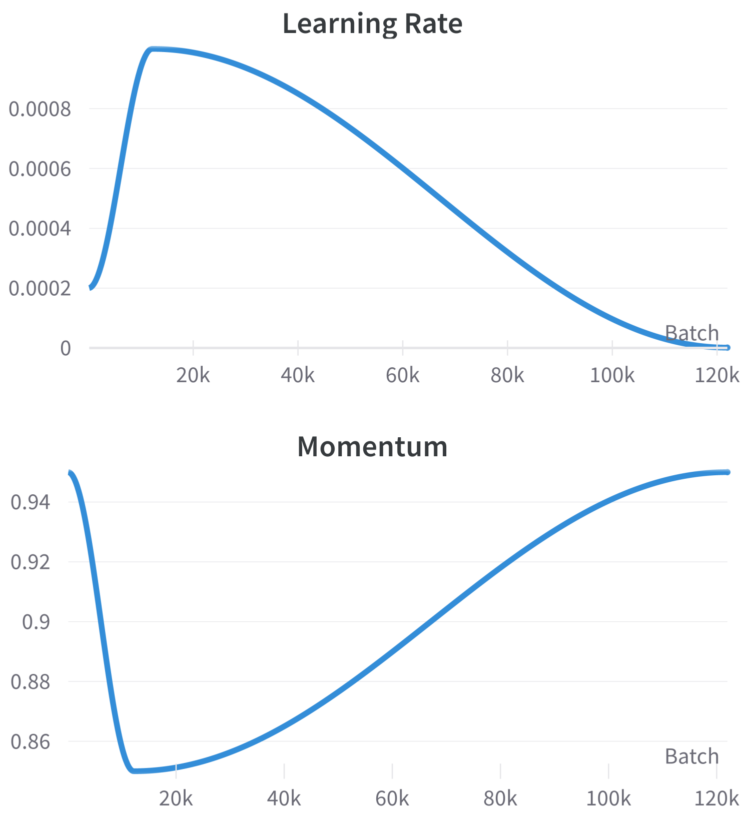

Figure 4.7: Learning rate and momentum scheduling. Learning rate and momentum are scheduled using cosine annealing [LH16] (see Eq. 4.12) according to the one-cycle policy [Smi18]. The base learning rate is set to $\alpha = 0.001$ and the base momentum is set to $\beta_1 = 0.95$.

Hyperparameters $\beta_2 = 0.99, \epsilon = 10^{-5}, \lambda = 0.01$ are constant during the training. β_1 and the learning rate α are scheduled during the training procedure using the one-cycle policy [Smi18] (see Figure 4.7) and cosine annealing [LH16] (see Eq. 4.12).

$$\eta_t = \eta^i_{min} + \frac{1}{2}(\eta^i_{max} - \eta^i_{min})(1 + \cos(\frac{T_{cur}}{T_i}\pi)) \tag{4.12}$$

"where η^i_{min} and η^i_{max} are ranges for the learning rate, and T_{cur} accounts for how many epochs have been performed since the last restart. Since T_{cur} is updated at each batch iteration t, it can take discredited values such as 0.1, 0.2, etc. Thus, $\eta_t = \eta^i_{max}$ when $t = 0$ and $T_{cur} = 0$. Once $T_{cur} = Ti$, the cos function will output -1 and thus $\eta_t = \eta^i_{min}$." [LH16]

The advantage of this approach is that it allows using a higher learning rate α adjusted by lowering the momentum β_1. The base learning rate is found using the algorithm by Smith [Smi17] that suggests gradually increasing the learning rate over batches, recording the loss until divergence and choosing the learning rate according to some policy, e.g. the tenth of the minimum.

The last important thing is that the learning rate and momentum differ across the layers of the network. The base learning rate in the first layers is smaller and increases gradually with deeper layers. The base momentum is adjusted accordingly (decreased with increasing learning rate). Fastai [HG20] natively supports these feature in all ResNet [HZRS16] models. Additionaly, it was added to all models, but NvGaze [KSM$^+$19a] where the hyperparameters were the same across the whole architecture.

The whole procedure was implemented using simple parameters and schedulers with the fastai [HG20] library. The parameters were slightly modified for the final model, when the architecture was selected.

4.1.4 Inference

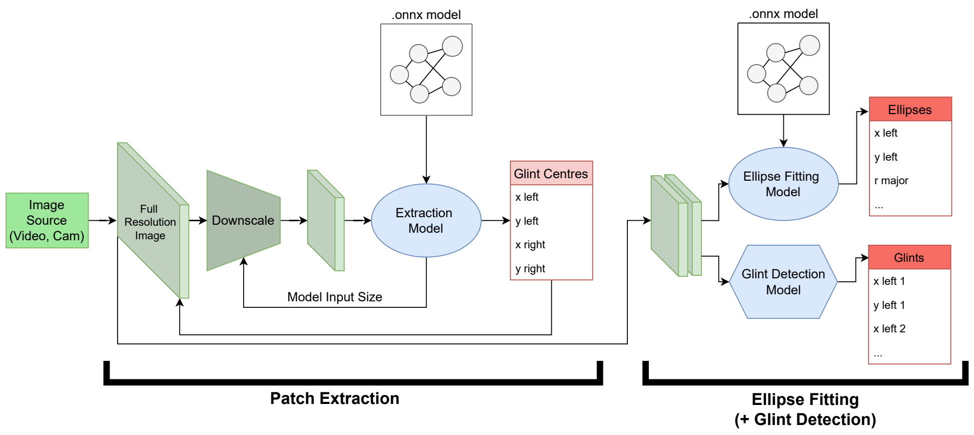

Figure 4.8: Inference pipeline. The input frames (from video or sequence of images) are downscaled to match the first model. The model estimates the pupil centres and extracts patches of size to match the second model. The images are stacked to form a batch, so the inference is slightly faster. The images are also fed to an image processing algorithm that locates glints.

Although the complete pipeline was tested only on the CPU using the OpenCV [Bra00] framework, the final model should run on hardware-specific optimized pipeline (see Section 6). Most of these pipelines, i.e. DeepStream [NVIa], OpenVINO [Int], accept neural network models in ONNX [dev21] or other standard format. A simple pipeline [Gil] was modified to accept the ONNX model, and with its modular design, it offered a great deal of variability in the pipeline debugging. The final pipeline is summed up in Figure 4.8.

As the figure suggests, the image is downscaled to the size accepted by the first ONNX model. The model outputs the predicted coordinates of the pupil centres. Two images of the size accepted by the second ONNX model are extracted around the pupil centres. The two images are fed to the second ONNX model according to its input shape, and an ellipse is fitted in each of them. In this work, the images are fed as a batch. The two extracted images are also used in the image processing algorithm that detects glints. These two models are independent and can be run parallelly.

4.2 Chosen model

Considering the comparison of different architectures (see 5.1) the NvGaze model [KSM+19a] (see Section 2.3) was chosen for the extraction task, as well as for the ellipse fitting (see List 3). Both models take augmented (see Section 4.1.1) grayscale images of size 293x293 as input with AdamW optimizer [LH17] with the same parameters as in Section 4.1.3. Hyperparameters were the same across the whole model architecture (unlike in ResNet, see Section 4.1.3).

4.2.1 Pupil extraction

The pupil extraction model (see Patch Extraction part in Figure 4.8) was trained for 1500 epochs with batch size 128 using the L2 loss function (see Eq. 4.2).

4.2.2 Ellipse fitting

The model for ellipse fitting (see Ellipse Fitting part in Figure 4.8) is extended with 3 linear (fully connected) layers with 512, 256 and 128 features. In this work, the model is referred to as NvGaze FC. This addition improved the accuracy of the model (see Section 5.1) at the expense of slightly higher inference time (see Section 5.2). The difference between NvGaze FC and NvGaze F2, which is the final model, is that NvGaze F2 was trained for 2000 (not 1500). Both models were trained with batch size 64. Also, in the training of NvGaze F2, the limit when the Shift, Scale, Rotate augmentation is discarded, considering the position of the pupil centre from the image border, was lowered. In other words, the pupils could be closer to the border.

4.2.3 Glint detection

The detection algorithm of the reflection of external infrared light sources, i.e. the glint detection was inspired by the work of Hosp et al. [HEM$^+$20].

At first, the image is pre-processed with common image processing techniques (see List 4.2.3 and Figure 4.9) using OpenCV [Bra00].

1. Denoise image with 3x3 Gaussian filter
2. Normalize image to range (0, 255)
3. Find edges with 3x3 2D-Laplacian filter
4. Normalize image to range (0, 255)
5. Clip values to 0 and 1 using binary threshold 30
6. Morphological close with 2x2 kernel of ones to close holes
7. Invert

Image 7 in Figure 4.9 shows the final pre-processed image. At this point, the algorithm finds contours and filters the irrelevant ones. The process is described in Algorithm 1. Note that the algorithm usually finds only one suitable pair, and the scoring procedure is skipped.

Since the glints are not annotated in the dataset, the method was tested visually on the video of the frames of the test dataset. The method works well if the glints are separated from each other as two individual disks without intersection. Otherwise the glints are not detected.

The method runs at 2.5ms on a CPU for one 293x293 image using OpenCV [Bra00] python. In the original paper, the method runs at 0.46ms for one 100x100 px image using OpenCV [Bra00] C/C++.

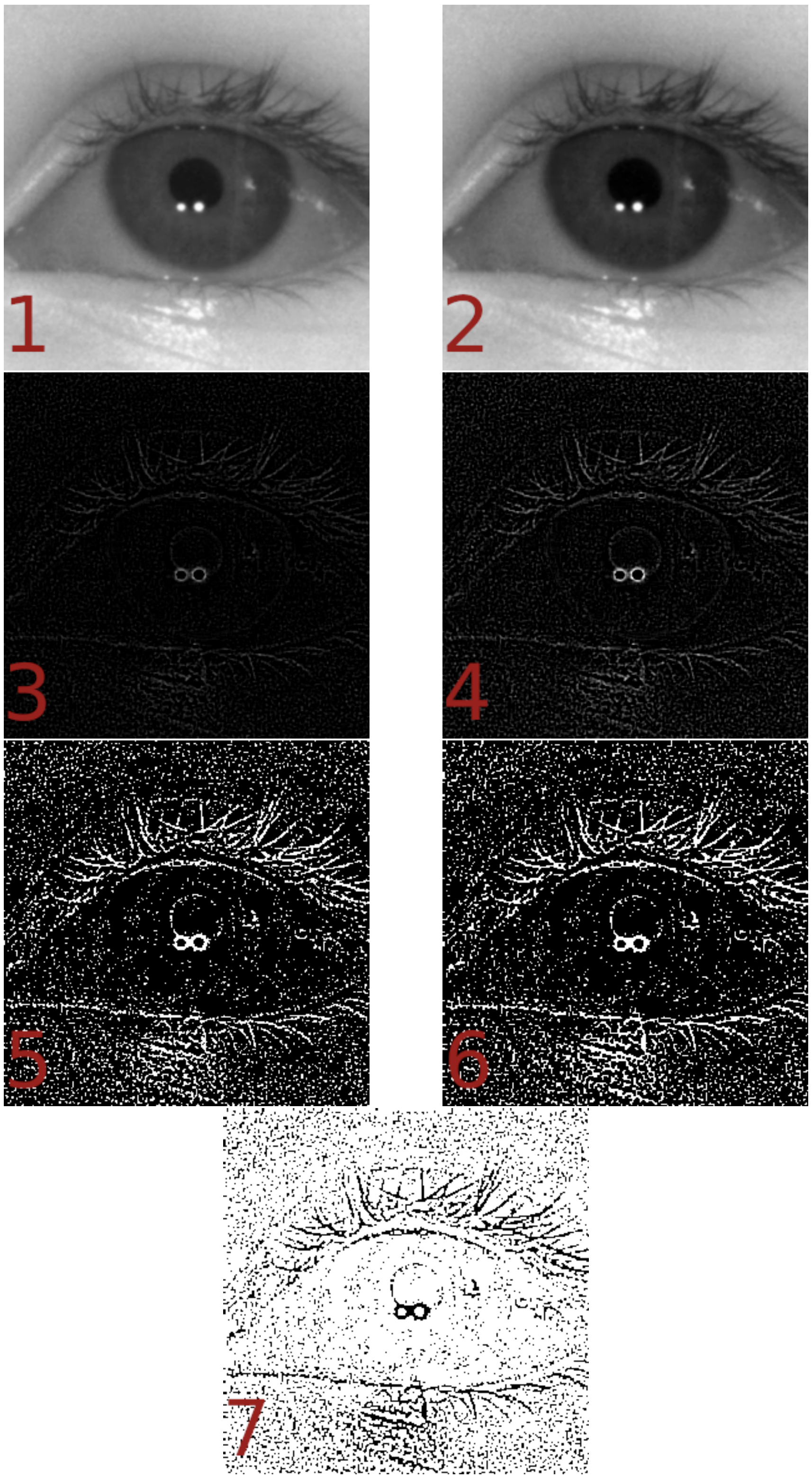

Figure 4.9: Glint detection image processing. From left to right, up to down, each image corresponds to a step in List 4.2.3.

Algorithm 1 Pseudocode of glint detection

```
Require: Binary Image $i$, Original denoised image $g$
 1: Find contours using simple chain approximation in $i$.
 2: $centres \leftarrow []$
 3: for all contours do
 4:     Calculate bounding box $bbox$
 5:     if width or height of $bbox$ is outside the range (5, 15) then
 6:         discard it
 7:     else
 8:         find its $centre$
 9:         $centres$.append($centre$)
10:     end if
11: end for
12: $pairs \leftarrow []$
13: Sort $centres$ by y
14: for all $centres$ but the last one do
15:     if the next centre in $centres$ is within 6 pixels then
16:         create a $pair$ from these two centres
17:         $pairs$.append($pair$)
18:     end if
19: end for
20: $scores \leftarrow []$
21: for all $pairs$ do
22:     if the distance between the two $centres$ in $pair$ is outside the range
    (6, 20) then
23:         skip it
24:     else
25:         Calculate brightness scores $b1$ and $b2$ from the 3x3 square with
    centre in each $centre$ from $g$ as the mean of all values in the square
26:         $b \leftarrow b1 + b2$
27:         Calculate shadow scores $s1$ and $s2$ from the 10x10 square with
    centre in each $centre$ excluding the 3x3 square from $g$ as the mean of all
    values in the square
28:         $s \leftarrow s1 + s2$
29:         $scores$.append($s$)
30:         save $b$ and $s$ to $pair$
31:     end if
32: end for
33: Sort $pairs$ according to $scores$ in ascending order
34: $backup \leftarrow$ the pair with minimum shadow score from $pairs$
35: for all $pairs$ do
36:     if $b \geq 300$ and $s \leq 200$ then
37:         return $pair$
38:     end if
39: end for
40: return $backup$
```

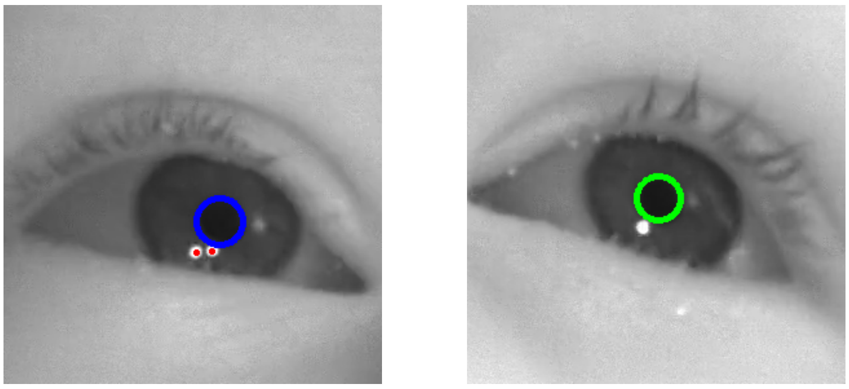

Figure 4.10: Glint detection examples. (left) Examples of correct glint localization. (right) Examples of incorrect glint localization.

In Figure 4.10, there is an example of correct and incorrect glint localization on the test dataset.

4.3 Evaluation description

It is essential to evaluate the trained networks in terms of both accuracy and speed. The following sections explain the motivation behind the two methods and describe them.

4.3.1 Detection Accuracy

In the task of **pupil patch extraction** (see List 3), the pupil centre should be estimated as accurately as possible. However, as the high-resolution image containing the whole face is downscaled, it becomes a very difficult task to find the centres correctly. For this reason, a square patch around the first estimation is extracted for the next stage, i.e. the ellipse fitting. The task is thus to estimate the pupil centre, such that the whole pupil is in the square patch. The metric that compares models from this group is the maximum L2 distance between the estimated and ground-truth pupil centre among all images in the test dataset (see Eq. 4.13).

$$E = \max(e_i) \quad i = 1, ..., n \tag{4.13}$$

where e_i is the L2 distance between the predicted and real pupil centre location of eye i and n denotes the number of eyes in the dataset.

Considering the task of **ellipse fitting** (see List 3), most of the above mentioned papers measured the accuracy by the pupil detection rate (see Figure 2.4). The x-pixel detection rate is described in the following way: "Accuracy of pupil estimation is usually given in form of a probability of estimating the pupil location with a maximum distance of x pixels from the ground truth pupil location" [FSK+17]. In this work 3-pixel and 5-pixel errors were measured on the given test dataset for each model.

$$f_x(e) = \begin{cases} 1 & \text{if } e \leq x \\ 0 & \text{otherwise} \end{cases} \tag{4.14}$$

$$E_x = \frac{1}{n}\sum_{i=1}^{n} f_x(e_i) \tag{4.15}$$

where e_i is the L2 distance between the predicted and real pupil centre location of eye i and n denotes the number of eyes in the dataset.

In case of the 5-pixel error that would be.

$$f_5(e) = \begin{cases} 1 & \text{if } e \leq 5 \\ 0 & \text{otherwise} \end{cases} \tag{4.16}$$

$$E_5 = \frac{1}{n}\sum_{i=1}^{n} f_5(e_i) \tag{4.17}$$

As this thesis focuses not only on detecting the centre of the pupil but also on fitting the ellipse around the pupil correctly, the error had to be adjusted to work with ellipses. Considering how the ellipse loss was defined (see Eq. 4.3 and 4.4. See also Figure 4.6), the ellipse can be considered as correctly detected if all of the 4 points defining the predicted ellipse are within x-pixels from their true position. The only difference in Eq. 4.17 is that e_i is the maximum of the 4 errors that occurred from the 4 points defining the ellipse (see Eq. 4.18). This metric will be further referred to as the ellipse detection rate.

$$e_i = \max(e_{ip}) \quad p = \{1, 2, 3, 4\} \tag{4.18}$$

4.3.2 FPS benchmark

As the neural networks are trained and exported to the .onnx format [dev21], they are ready to be benchmarked. The purpose of the benchmark is to compare neural network architectures in terms of speed without extra operations, e.g. copying tensors from CPU to GPU. In this way, the networks can be compared against each other in terms of pure inference time. Contrary, the inference time in production will differ from the measured inference time. However, as can be seen from the results (see Figure 5.2), the inference time can differ by a factor of 3, depending on the GPU. The benchmark runs on Google Colab [Goo] using jupyter notebooks. In this manner it can be replicated.

The framework allows to run inference on a few different GPU units that are assigned to the user. In this work, the networks were benchmarked on 2 NVIDIA GPUs: Tesla T4 [NVIc] and Tesla K80 [NVIb].

The test begins by obtaining a GPU unit. Then random data and a model are transferred to the GPU for a session of 1000 inference batches. First 5 batches are treated as warm-up batches and are discarded. From the remaining 995 batches, the median is taken. Each model then runs for 5 sessions. Medians of the 5 sessions are averaged, and the value p called processing time is obtained. The value is inverted $\text{fps} = \frac{1}{p}$ to retrieve frames per second and used for the final comparison.

Figure 4.11 shows that the first batches of a benchmark fps are significantly slower than the rest of the batches. For this reason, the first 5 batches are discarded and marked as warm-up batches.

The rest of the batches still contains outliers, i.e. runs where the inference time was much higher than in the rest of the runs. Thus the median value of each session is calculated as the representative of each session (see Figure 4.12).

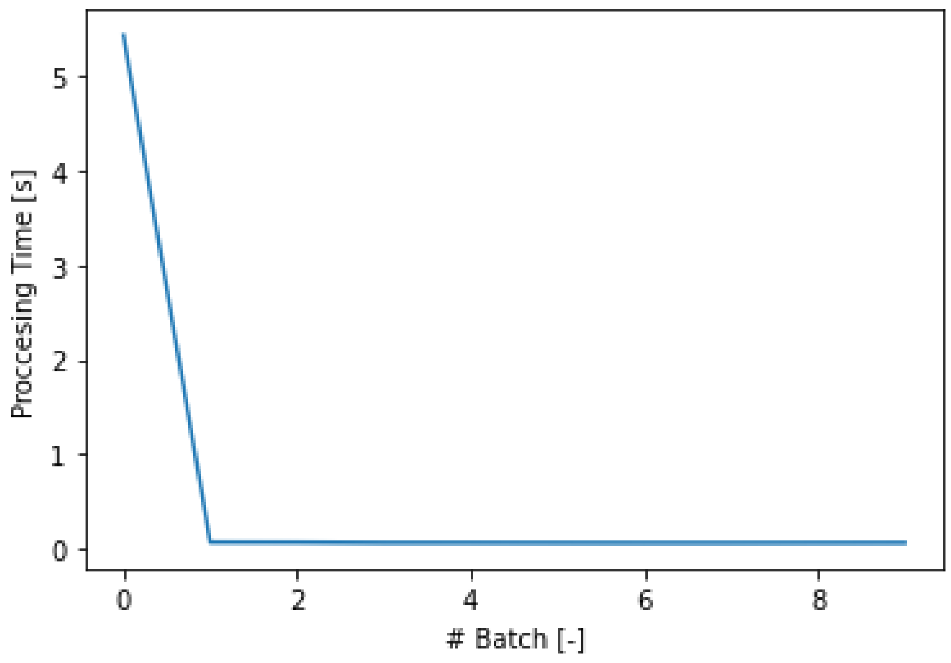

Figure 4.11: Benchmark warm-up. The reason for benchmark warm-up is that the initial epochs of each benchmark are significantly slower than the rest of them.

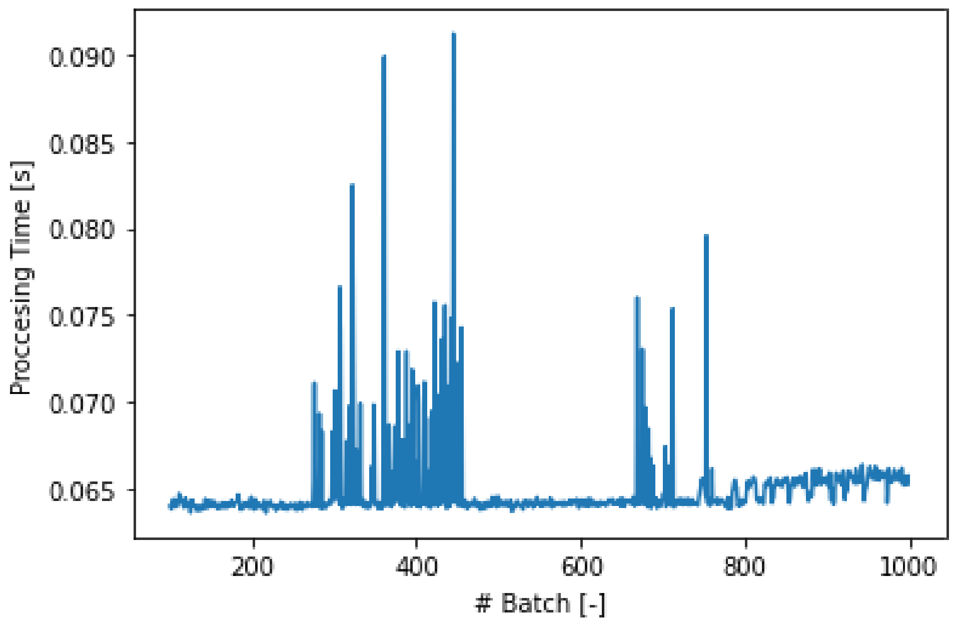

Figure 4.12: Benchmark median. When the warm-up episodes are discarded, the rest of the measured times contains still many outliers. For this reason, median is calculated from the values, which is known to be less sensitive to outliers.

Chapter 5

Results

Considering the image extraction task and the ellipse fitting task from List 3, several neural network architectures were tested in terms of accuracy and inference time. In this chapter, I will present the results of the comparison, that led to the choice of the final fine-tuned architectures presented in Section 4.2. The details of the neural networks are presented throughout Section 4.1.

The next section presents the results of the detection accuracy (see Section 4.3.1) of the above-mentioned networks. The following section presents the FPS benchmark (see Section 4.3.2) of those networks, with some extra variations of those architectures for reference that were not trained.

5.1 Detection Accuracy

In Figure 5.1, the architectures from the patch extraction task are compared using the maximum L2 distance on the test dataset as proposed in Section 4.3.1.

In Section 4.3.1, the metrics called detection rate and ellipse detection rate are introduced. The purpose of these metrics is to calculate how many predictions on the images from the test dataset are within x pixels, i.e. 5 pixels, within the ground truth pupil position. In Figure 5.2, the resulting detection rates of the architectures from Table 4.2 are presented.

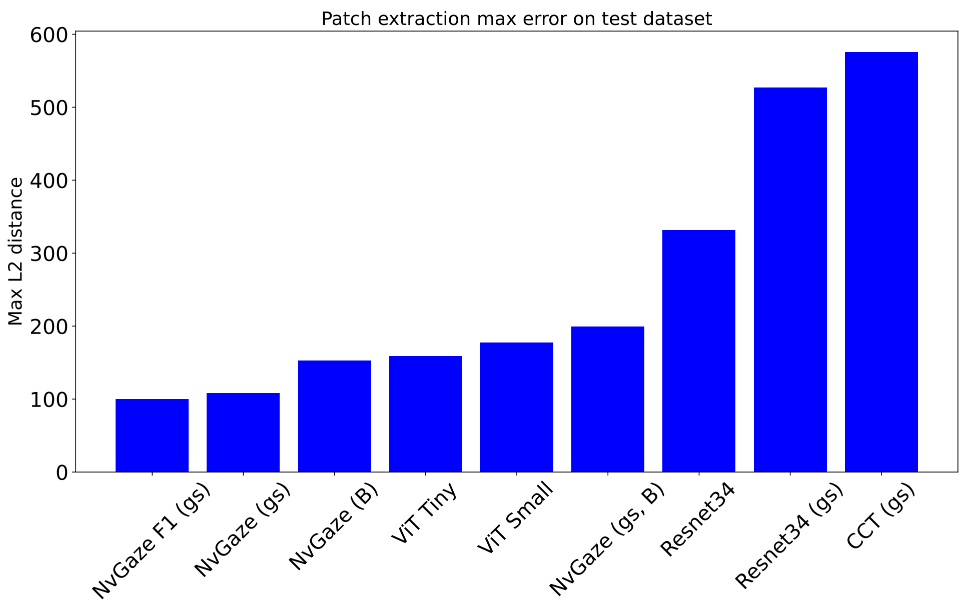

Figure 5.1: Patch extraction accuracy results. The maximum L2 distances on the test dataset of the neural network architectures from Table 4.1 and the final fine-tuned model NvGaze F1 (see Section 4.2.1).

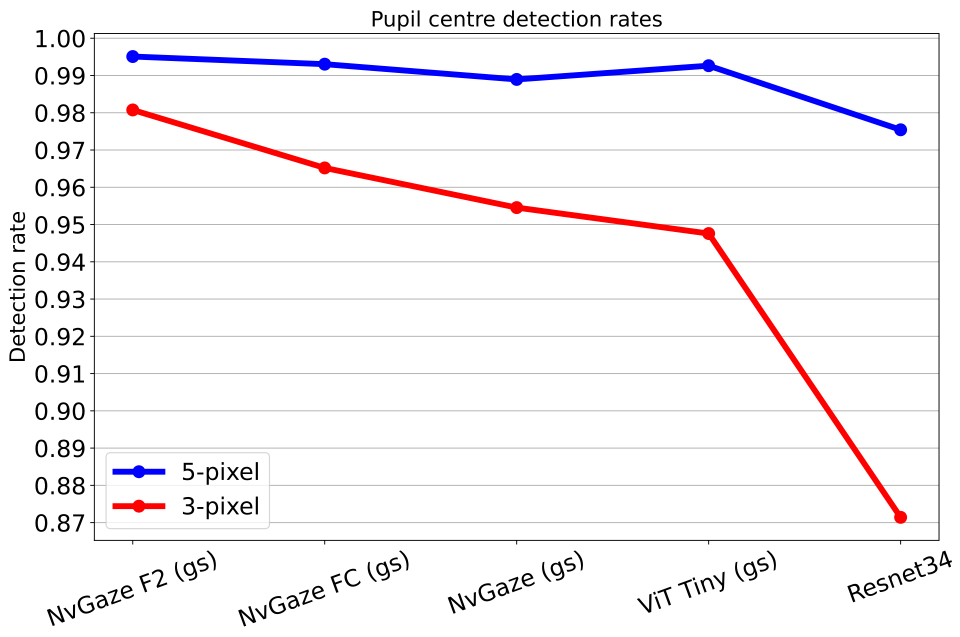

Figure 5.2: Detection rates of the architectures considered for the ellipse fitting task. The 5-pixel and 3-pixel detection rates of the neural network architectures from Table 4.2 and the final fine-tuned model NvGaze F2 (see Section 4.2.2).

To work with ellipses, the detection rate was adjusted so that each of the 4 points defining the ellipse are within x pixels of the ground truth position.

The ellipse detection rate of the same networks is presented in Figure 5.3.

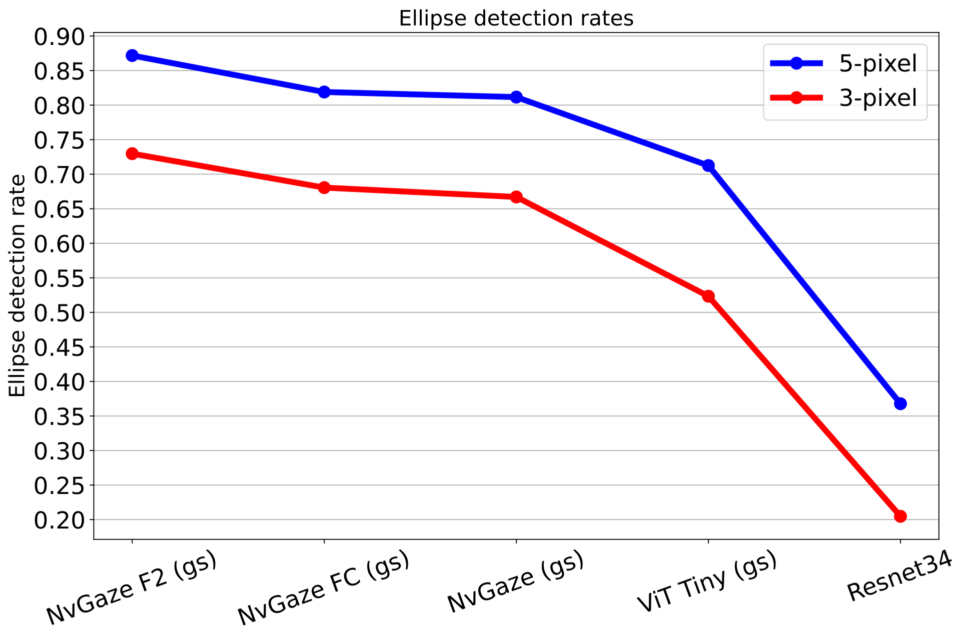

Figure 5.3: Ellipse detection rates of the architectures considered for the ellipse fitting task. The 5-pixel and 3-pixel ellipse detection rates of the neural network architectures from Table 4.2 and the final fine-tuned model NvGaze F2 (see Section 4.2.2).

5.2 FPS benchmark

The benchmark procedure was described in Section 5.2. Note that some architectures that were benchmarked are not presented in the results, as they were not trained. Those are marked with the asterisk (*) symbol. The following figure compares the frames per second rates of the networks used for pupil extraction (see Figure 5.4).

The networks considered for ellipse fitting yielded the following results (see Figure 5.5).

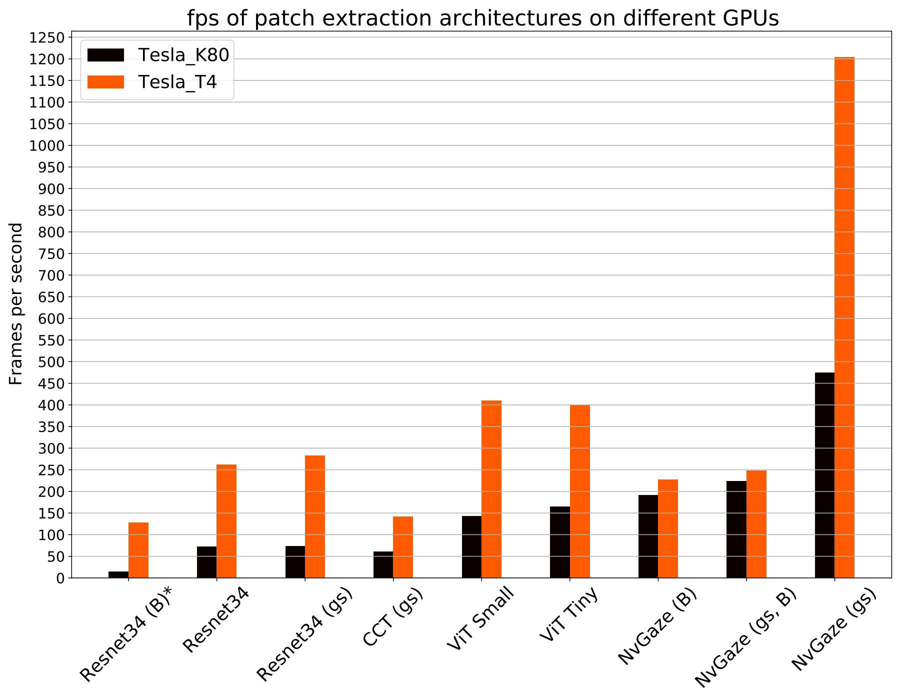

Figure 5.4: Patch extraction benchmark results. The frames per second rates of the neural network architectures in Table 4.1 and their modifications that were not trained denoted with the asterisk (*) symbol.

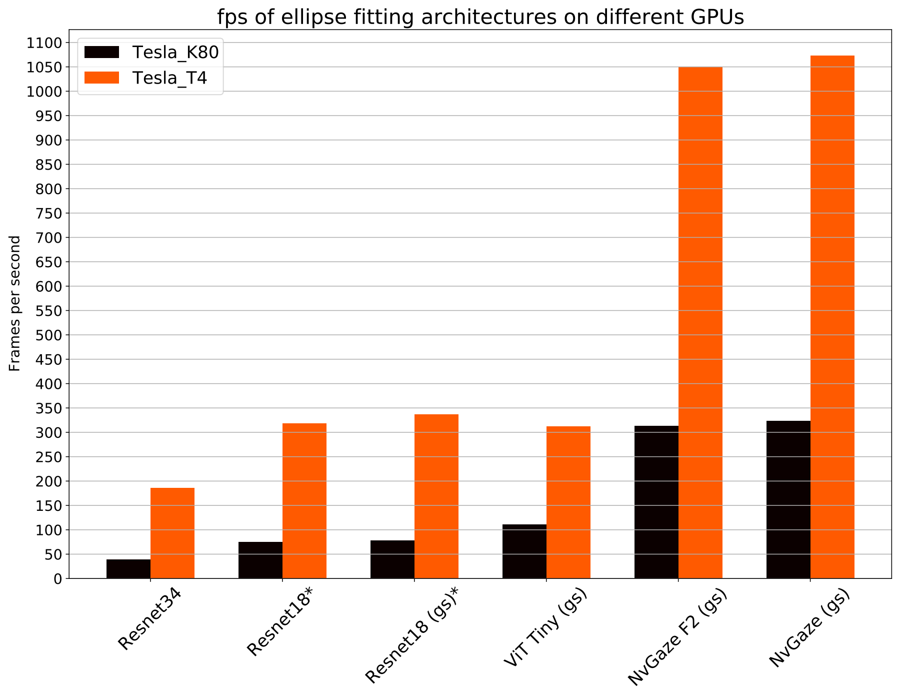

Figure 5.5: Ellipse fitting benchmark results. The frames per second rates of the neural network architectures in Table 4.2, their modifications that were not trained denoted with the asterisk (*) symbol and the final fine-tuned model NvGaze F2.

Chapter 6

Discussion and future work

The main goal of this thesis was to build a framework that is suitable for the continuous development of models used in eye-tracking. Certainly, the main features of the proposed framework, presented in Section 4.1, are the ease of integration of novel neural network architectures and techniques, the variety of datasets for these models, and tracking and comparison of the models through the state of the art frameworks.

The framework was tested on an eye-tracking dataset provided by the supervisor of this thesis. Several architectures were researched and considered, including Resnet [HZRS16], ViT [DBK+20] and NvGaze [KSM+19a]. The comparison of these models in terms of accuracy and inference time was presented in Chapter 5. As can be seen from the results, the NvGaze model [KSM+19a], with its simple architecture to be fast, outperformed the other models in terms of speed by a great amount (see Section 5.2). Considering the tested architectures, the model also outperformed other models in terms of accuracy 5.1.

As our goal was to test the framework with several models and choose the one that works best for the provided dataset, the network was not trained on any popular dataset [KSM+19a] [FSK+17]. The downside of this approach is that models that require a huge amount of data, i.e. vision transformers [DBK+20], may perform better when trained on a different dataset. Conversely, the advantage of this strategy is that models converge faster on smaller datasets, and all of the networks could be trained within a day. In addition, the role of dataset augmentation is much more significant on smaller datasets, and the most relevant augmentations could be chosen.

That said, the NvGaze architecture was chosen for this thesis mainly due to its simplicity and possibility to run the model on cheaper GPUs yet still achieving real-time performance. To demonstrate the modularity of trained and exported .onnx networks [dev21], the networks were tested on a simple inference pipeline (see Section 4.1.4).

In addition to the possible improvement of all the networks, especially vision transformers [DBK+20], through bigger datasets, it is essential to consider segmentation datasets as well. As was described in Section 2.2, the RitNet network [CKA+19] showed promising results on the OpenEDS challenge [GSS+19]. For this reason, the main point of future work is the integration of the segmentation task in the framework. In parallel, the regression task can be extended with the popular datasets in order to compare the trained models to the existing ones.

The other important field of future work is building a hardware-specific inference pipeline such as Deepstream [NVIa] or OpenVINO [Int] to integrate the trained .onnx models [dev21]. This part can be designed separately from the framework.

Chapter 7

Conclusion

This work proposes a framework for the development of eye-tracking models that integrates state of the art libraries. The framework integrates novel neural network architectures and techniques for several instances of a problem of an eye-tracking pipeline. The framework was tested on a dataset, provided by the supervisor of this thesis, with numerous neural network architectures. After testing the most suitable model in terms of accuracy and real-time performance, the model was further fine-tuned to achieve better accuracy on the test dataset. By testing the framework on the real eye-tracking problem, the thesis verified that the framework can be used in the continuous development of eye-tracking models that are based on novel neural network architectures.

Bibliography

[ABK18] E. Khvedchenya V. I. Iglovikov A. Buslaev, A. Parinov and A. A. Kalinin, *Albumentations: fast and flexible image augmentations*, ArXiv e-prints (2018).

[Bie20] Lukas Biewald, *Experiment tracking with weights and biases*, 2020, Software available from wandb.com.

[Bra00] G. Bradski, *The OpenCV Library*, Dr. Dobb's Journal of Software Tools (2000).

[Can86] John Canny, *A computational approach to edge detection*, IEEE Transactions on pattern analysis and machine intelligence (1986), no. 6, 679–698.

[CC21] Alex Clark and Contributors, *Python imaging library (pillow fork)*, `https://pillow.readthedocs.io/en/stable/#`, 2021.

[CK07] Yen-Wei Chen and Kenji Kubo, *A robust eye detection and tracking technique using gabor filters*, Third International Conference on Intelligent Information Hiding and Multimedia Signal Processing (IIH-MSP 2007), vol. 1, IEEE, 2007, pp. 109–112.

[CKA$^+$19] Aayush K Chaudhary, Rakshit Kothari, Manoj Acharya, Shusil Dangi, Nitinraj Nair, Reynold Bailey, Christopher Kanan, Gabriel Diaz, and Jeff B Pelz, *Ritnet: Real-time semantic segmentation of the eye for gaze tracking*, 2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW), IEEE, 2019, pp. 3698–3702.

[CKK19] Viviane Clay, Peter König, and Sabine Koenig, *Eye tracking in virtual reality*, Journal of eye movement research **12** (2019), no. 1.

[DBK+20] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby, *An image is worth 16x16 words: Transformers for image recognition at scale*, 2020.

[dev21] ONNX Runtime developers, *Onnx runtime*, `https://onnxruntime.ai/`, 2021.

[DKB18] Kai Dierkes, Moritz Kassner, and Andreas Bulling, *A novel approach to single camera, glint-free 3d eye model fitting including corneal refraction*, Proceedings of the 2018 ACM Symposium on Eye Tracking Research & Applications, 2018, pp. 1–9.

[ESK+19] Shaharam Eivazi, Thiago Santini, Alireza Keshavarzi, Thomas Kübler, and Andrea Mazzei, *Improving real-time cnn-based pupil detection through domain-specific data augmentation*, Proceedings of the 11th ACM Symposium on Eye Tracking Research & Applications, 2019, pp. 1–6.

[FB81] Martin A Fischler and Robert C Bolles, *Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography*, Communications of the ACM **24** (1981), no. 6, 381–395.

[FEH+18] Wolfgang Fuhl, Shahram Eivazi, Benedikt Hosp, Anna Eivazi, Wolfgang Rosenstiel, and Enkelejda Kasneci, *Bore: Boosted-oriented edge optimization for robust, real time remote pupil center detection*, Proceedings of the 2018 ACM Symposium on Eye Tracking Research & Applications, 2018, pp. 1–5.

[FGS+18] Wolfgang Fuhl, David Geisler, Thiago Santini, Tobias Appel, Wolfgang Rosenstiel, and Enkelejda Kasneci, *Cbf: Circular binary features for robust and real-time pupil center detection*, Proceedings of the 2018 ACM symposium on eye tracking research & applications, 2018, pp. 1–6.

[FKS+15] Wolfgang Fuhl, Thomas Kübler, Katrin Sippel, Wolfgang Rosenstiel, and Enkelejda Kasneci, *Excuse: Robust pupil detection in real-world scenarios*, International conference on computer analysis of images and patterns, Springer, 2015, pp. 39–51.

[FSK+17] Wolfgang Fuhl, Thiago Santini, Gjergji Kasneci, Wolfgang Rosenstiel, and Enkelejda Kasneci, *Pupilnet v2. 0: Convolutional neural networks for cpu based real time robust pupil detection*, arXiv preprint arXiv:1711.00112 (2017).

[FSKK16] Wolfgang Fuhl, Thiago C Santini, Thomas Kübler, and Enkelejda Kasneci, *Else: Ellipse selection for robust pupil detection in real-world environments*, Proceedings of the Ninth Biennial ACM Symposium on Eye Tracking Research & Applications, 2016, pp. 123–130.

[GDG$^+$17] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He, *Accurate, large minibatch sgd: Training imagenet in 1 hour*, arXiv preprint arXiv:1706.02677 (2017).

[GE06] Elias Daniel Guestrin and Moshe Eizenman, *General theory of remote gaze estimation using the pupil center and corneal reflections*, IEEE Transactions on biomedical engineering **53** (2006), no. 6, 1124–1133.

[Gil] Jarosław Gilewski, *Image processing pipeline.*

[Goo] Google, *Google colaboratory.*

[GSS$^+$19] Stephan J Garbin, Yiru Shen, Immo Schuetz, Robert Cavin, Gregory Hughes, and Sachin S Talathi, *Openeds: Open eye dataset*, arXiv preprint arXiv:1905.03702 (2019).

[HEM$^+$20] Benedikt Hosp, Shahram Eivazi, Maximilian Maurer, Wolfgang Fuhl, David Geisler, and Enkelejda Kasneci, *Remoteeye: An open-source high-speed remote eye tracker*, Behavior research methods **52** (2020), no. 3, 1387–1401.

[HG20] Jeremy Howard and Sylvain Gugger, *Fastai: a layered api for deep learning*, Information **11** (2020), no. 2, 108.

[HHM$^+$18] Erik Hernández, Santiago Hernández, David Molina, Rafael Acebrón, and Cecilia E García Cena, *Oscann: Technical characterization of a novel gaze tracking analyzer*, Sensors **18** (2018), no. 2, 522.

[HJ09] Dan Witzner Hansen and Qiang Ji, *In the eye of the beholder: A survey of models for eyes and gaze*, IEEE transactions on pattern analysis and machine intelligence **32** (2009), no. 3, 478–500.

[HK18] Katarzyna Harezlak and Pawel Kasprowski, *Application of eye tracking in medicine: A survey, research issues and challenges*, Computerized Medical Imaging and Graphics **65** (2018), 176–190.

[HMvdW$^+$20] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant, *Array programming with NumPy*, Nature **585** (2020), no. 7825, 357–362.

[How] Jeremy Howard, *imagenette*.

[HWS+21] Ali Hassani, Steven Walton, Nikhil Shah, Abulikemu Abuduweili, Jiachen Li, and Humphrey Shi, *Escaping the big data paradigm with compact transformers*, arXiv preprint arXiv:2104.05704 (2021).

[HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, *Delving deep into rectifiers: Surpassing human-level performance on imagenet classification*, Proceedings of the IEEE international conference on computer vision, 2015, pp. 1026–1034.

[HZRS16] ______, *Deep residual learning for image recognition*, Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 770–778.

[Int] Intel, *Openvino*.

[JHB+15] Amir-Homayoun Javadi, Zahra Hakimi, Morteza Barati, Vincent Walsh, and Lili Tcheang, *Set: a pupil detection method using sinusoidal approximation*, Frontiers in neuroengineering **8** (2015), 4.

[KB14] Diederik P Kingma and Jimmy Ba, *Adam: A method for stochastic optimization*, arXiv preprint arXiv:1412.6980 (2014).

[KCB+21] Rakshit S Kothari, Aayush K Chaudhary, Reynold J Bailey, Jeff B Pelz, and Gabriel J Diaz, *Ellseg: An ellipse segmentation framework for robust gaze tracking*, IEEE Transactions on Visualization and Computer Graphics **27** (2021), no. 5, 2757–2767.

[KKK+16] Kyle Krafka, Aditya Khosla, Petr Kellnhofer, Harini Kannan, Suchendra Bhandarkar, Wojciech Matusik, and Antonio Torralba, *Eye tracking for everyone*, Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 2176–2184.

[KMN+16] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang, *On large-batch training for deep learning: Generalization gap and sharp minima*, arXiv preprint arXiv:1609.04836 (2016).

[KPB14] Moritz Kassner, William Patera, and Andreas Bulling, *Pupil: an open source platform for pervasive eye tracking and mobile gaze-based interaction*, Proceedings of the 2014 ACM international joint conference on pervasive and ubiquitous computing: Adjunct publication, 2014, pp. 1151–1160.

[KSM+19a] Joohwan Kim, Michael Stengel, Alexander Majercik, Shalini De Mello, David Dunn, Samuli Laine, Morgan McGuire, and David Luebke, *Nvgaze: An anatomically-informed dataset for low-latency, near-eye gaze estimation*, Proceedings of the 2019 CHI conference on human factors in computing systems, 2019, pp. 1–12.

[KSM+19b] ———, *Nvgaze: An anatomically-informed dataset for low-latency, near-eye gaze estimation supplementary material*, 2019, pp. 1–20.

[LB+95] Yann LeCun, Yoshua Bengio, et al., *Convolutional networks for images, speech, and time series*, The handbook of brain theory and neural networks **3361** (1995), no. 10, 1995.

[LCM19] Agostina J Larrazabal, CE García Cena, and César Ernesto Martínez, *Video-oculography eye tracking towards clinical applications: A review*, Computers in biology and medicine **108** (2019), 57–66.

[LG68] Yves Le Grand, *Light, colour and vision*, Chapman & Hall, 1968.

[LH16] Ilya Loshchilov and Frank Hutter, *Sgdr: Stochastic gradient descent with warm restarts*, arXiv preprint arXiv:1608.03983 (2016).

[LH17] ———, *Decoupled weight decay regularization*, arXiv preprint arXiv:1711.05101 (2017).

[LWP05] Dongheng Li, David Winfield, and Derrick J Parkhurst, *Starburst: A hybrid algorithm for video-based eye tracking combining feature-based and model-based approaches*, 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)-Workshops, IEEE, 2005, pp. 79–79.

[MEK+01] Michael Mrochen, Mostafa Salah Eldine, Maik Kaemmerer, Theo Seiler, and Werner Hütz, *Improvement in photorefractive corneal laser surgery results using an active eye-tracking system*, Journal of Cataract & Refractive Surgery **27** (2001), no. 7, 1000–1006.

[MF12] Maria Laura Mele and Stefano Federici, *Gaze and eye-tracking solutions for psychological research*, Cognitive processing **13** (2012), no. 1, 261–265.

[NVIa] NVIDIA, *Deepstream sdk.*

[NVIb] ______, *Nvidia tesla k80.*

[NVIc] ______, *Nvidia tesla t4.*

[PGM+19] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala, *Pytorch: An imperative style, high-performance deep learning library*, Advances in Neural Information Processing Systems 32 (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), Curran Associates, Inc., 2019, pp. 8024–8035.

[PY09] Sinno Jialin Pan and Qiang Yang, *A survey on transfer learning*, IEEE Transactions on knowledge and data engineering **22** (2009), no. 10, 1345–1359.

[PZBH18] Seonwook Park, Xucong Zhang, Andreas Bulling, and Otmar Hilliges, *Learning to find eye region landmarks for remote gaze estimation in unconstrained settings*, Proceedings of the 2018 ACM symposium on eye tracking research & applications, 2018, pp. 1–10.

[RFB15] Olaf Ronneberger, Philipp Fischer, and Thomas Brox, *U-net: Convolutional networks for biomedical image segmentation*, International Conference on Medical image computing and computer-assisted intervention, Springer, 2015, pp. 234–241.

[RHW86] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams, *Learning representations by back-propagating errors*, nature **323** (1986), no. 6088, 533–536.

[Sax18] Ujjwal Saxena, *Automold road augmentation library*, 2018.

[ŚBD12] Lech Świrski, Andreas Bulling, and Neil Dodgson, *Robust real-time pupil tracking in highly off-axis images*, Proceedings of the symposium on eye tracking research and applications, 2012, pp. 173–176.

[SD13] Lech Swirski and Neil Dodgson, *A fully-automatic, temporal approach to single camera, glint-free 3d eye model fitting*, Proc. PETMEI (2013), 1–11.

[SFK18] Thiago Santini, Wolfgang Fuhl, and Enkelejda Kasneci, *Pure: Robust pupil detection for real-time pervasive eye tracking*, Computer Vision and Image Understanding **170** (2018), 40–50.

[SK10] Weston Sewell and Oleg Komogortsev, *Real-time eye gaze tracking with an unmodified commodity webcam employing a neural network*, CHI'10 Extended Abstracts on Human Factors in Computing Systems, 2010, pp. 3739–3744.

[SLCLV20] Jose Luis Sanchez-Lopez, Manuel Castillo-Lopez, and Holger Voos, *Semantic situation awareness of ellipse shapes via deep learning for multirotor aerial robots with a 2d lidar*, 2020 International Conference on Unmanned Aircraft Systems (ICUAS), IEEE, 2020, pp. 1014–1023.

[SLJ$^+$15] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich, *Going deeper with convolutions*, Proceedings of the IEEE conference on computer vision and pattern recognition, 2015, pp. 1–9.

[Smi17] Leslie N Smith, *Cyclical learning rates for training neural networks*, 2017 IEEE winter conference on applications of computer vision (WACV), IEEE, 2017, pp. 464–472.

[Smi18] ______, *A disciplined approach to neural network hyper-parameters: Part 1–learning rate, batch size, momentum, and weight decay*, arXiv preprint arXiv:1803.09820 (2018).

[TKA02] Kar-Han Tan, David J Kriegman, and Narendra Ahuja, *Appearance-based eye gaze estimation*, Sixth IEEE Workshop on Applications of Computer Vision, 2002.(WACV 2002). Proceedings., IEEE, 2002, pp. 191–195.

[TKC00] Ying-li Tian, Takeo Kanade, and Jeffrey F Cohn, *Eye-state action unit detection by gabor wavelets*, International Conference on Multimodal Interfaces, Springer, 2000, pp. 143–150.

[TS09] Matthew E Taylor and Peter Stone, *Transfer learning for reinforcement learning domains: A survey.*, Journal of Machine Learning Research **10** (2009), no. 7.

[VJ01] Paul Viola and Michael Jones, *Rapid object detection using a boosted cascade of simple features*, Proceedings of the 2001 IEEE computer society conference on computer vision and pattern recognition. CVPR 2001, vol. 1, Ieee, 2001, pp. I–I.

[Wan21] P. Wang, *vit-pytorch*, 2021.

[Wig19] Ross Wightman, *Pytorch image models*, `https://github.com/rwightman/pytorch-image-models`, 2019.

[WP08] Michel Wedel and Rik Pieters, *Eye tracking for visual marketing*, Now Publishers Inc, 2008.

[WRvA+19] Zhengyang Wu, Srivignesh Rajendran, Tarrence van As, Joelle Zimmermann, Vijay Badrinarayanan, and Andrew Rabinovich, *Eyenet: A multi-task network for off-axis eye gaze estimation and user understanding*, ArXiv **abs/1908.09060** (2019).

[WRvA+20] Zhengyang Wu, Srivignesh Rajendran, Tarrence van As, Joelle Zimmermann, Vijay Badrinarayanan, and Andrew Rabinovich, *Magiceyes: A large scale eye gaze estimation dataset for mixed reality*, arXiv preprint arXiv:2003.08806 (2020).

[WZS16] Jianzhong Wang, Guangyue Zhang, and Jiadong Shi, *2d gaze estimation based on pupil-glint vector using an artificial neural network*, Applied Sciences **6** (2016), no. 6, 174.

[YAR+19] Yuk-Hoi Yiu, Moustafa Aboulatta, Theresa Raiser, Leoni Ophey, Virginia L Flanagin, Peter Zu Eulenburg, and Seyed-Ahmad Ahmadi, *Deepvog: Open-source pupil segmentation and gaze estimation in neuroscience using deep learning*, Journal of neuroscience methods **324** (2019), 108307.

[ZPB+20] Xucong Zhang, Seonwook Park, Thabo Beeler, Derek Bradley, Siyu Tang, and Otmar Hilliges, *Eth-xgaze: A large scale dataset for gaze estimation under extreme head pose and gaze variation*, European Conference on Computer Vision, Springer, 2020, pp. 365–381.

Appendix A

Contents of the enclosed CD

- eye-tracking – Library that contains the code described in this thesis