BABEŞ-BOLYAI UNIVERSITY FACULTY OF
MATHEMATICS AND COMPUTER SCIENCE
COMPUTER SCIENCE SPECIALIZATION

DIPLOMA THESIS

Eye-Tracking as an Intelligent Human-Computer Interface

Supervisor
Prof. Laura Silvia Dioșan

Author
Andrei-Paul Bejan

2024

UNIVERSITATEA BABEŞ-BOLYAI
FACULTATEA DE MATEMATICĂ ŞI INFORMATICĂ
SPECIALIZAREA INFORMATICĂ

LUCRARE DE LICENŢĂ

Interfață inteligentă între om și calculator bazată pe urmărirea privirii

Coordonator ştiinţific
Prof. Laura Silvia Dioșan

Absolvent
Andrei-Paul Bejan

2024

ABSTRACT

Current human-computer interfaces (e.g. mouse, trackpad, or keyboard) have gained in popularity over the years because they offered the simplest solution at the right time while having a reasonable learning curve. The technology has evolved rapidly in the last decades, and it is time to move towards new, intelligent, more intuitive, and faster human-computer interfaces. Eye-Tracking represents a promising step in this direction. Recent developments in appearance-based gaze estimation using deep learning algorithms have come a long way, and soon they will reach a consumer-ready performance level. This piece of work is intended to add another brick to the wall by proposing FastSightNet, a new efficient model based on the MobileNet architecture, evaluating it, and comparing it to existing models. My model is trained on the widely used MPIIFaceGaze dataset, achieving 5.1° mean angular error during leave-one-subject-out cross-validation. I also present GazeTrack, a system that incorporates the models and enables users to evaluate their performance in real-time on the webcam feed in both 3D world and 2D screen coordinates.

This piece of work starts off with the problem definition, followed by an extensive literature review on human-computer interfaces and eye-tracking methods. The Deep Learning chapter serves as a terminology foundation for the next chapter, where I explore the effectiveness of different types of deep architectures for the gaze estimation task. Chapter 6 introduces my GazeTrack system. Finally, I present another use-case for the Eye-Tracking model I developed, to measure Eye-Contact during a conversation.

The results of this work were disseminated at the Young Talents International Conference (YTIC) in St. Pölten, Austria.

Contents

Chapter 1

Introduction

The scope of this opening section is to present the problem I am approaching, the motivation and the objectives behind this piece of work as well as accurately situating the work among the broaden research body.

Context

Human-computer interface is a subject that has been studied extensively over the years, but has yet to become exhausted. The current most used human-computer interfaces such as mouse and keyboard in their various forms do a good job enabling people to control their computers but they are not the most intuitive, nor the fastest out there. One action humans perform unconsciously while navigating on their computer is to look directly at the component they want to press or select on the screen. Naturally, I arrive at one challenging problem, but very promising idea which is Eye-Tracking.

Eye-Tracking may sound like an impossible problem to solve algorithmically, since it is often unconstrained and it is really hard to create a model that can generalize well across multiple subjects. However, the last decade has shown an accelerated expansion in the machine learning field, culminating with deep architectures that can in fact learn very complex patterns from data and generalize well to unseen data.

Problem

The issue I am trying to solve is that current human-computer interfaces can be more intuitive, more efficient, and better suited for humans. The scope of this piece of work is to accelerate the development towards new human-computer interfaces based on Eye-Tracking technology. By providing a solution that works well in real-

life scenarios, everyone will benefit, and eventually, we can transition from traditional human-computer interfaces that merely accomplish tasks to innovative, more intuitive, and faster ways of interacting with computers.

Motivation

One of the reasons I am trying to solve this problem is that current human-computer interfaces have poor accessibility for people with upper-limb disabilities. Besides this, I consider the trackpad we usually find on laptops to be hard and very slow to use, while a periferic mouse is another object that people must carry around. Actually, in most real-life scenarios such as web browsing, reading text files, writing emails, and so on, an eye-tracking solution might be a better option, giving up on all the disadvantages of the aforementioned.

Objectives

The key objectives of this scientific work are threefold:

1. Reproduce literature state-of-the-art deep learning models
2. Propose a different method with the scope of achieving better performance
3. Create an application that can showcase the model performance and it's applicability in the aforementioned use-cases

Own contribution

The products of my own work are multiple:

1. Comprehensive literature review of existing human-computer interfaces and existing Eye-Tracking methods
2. Creation of the training pipeline based on custom configuration file
3. Integration of the training pipeline with the Weights and Biases API [2]
4. Comparison of the obtained results with other methods from the literature
5. Develop an application that enables users to preview real-time performance of any model in both 3D and 2D scenarios

Thesis structure

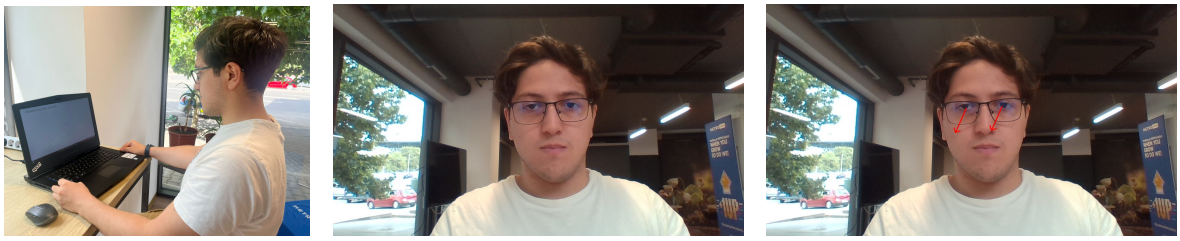
I will begin by describing the problem in a scientific manner in chapter 2, then I will present the background you need in terms of human-computer interfaces, with an accent on eye-tracking. I will introduce the most common datasets and methods that have been proposed in chapter 3. In chapter 4, I will follow with a description of each machine learning tool used in the upcoming chapters.

Section 5.3 presents the state-of-the-art model I have reproduced, trained myself, evaluated, and used as a baseline moving onwards. Section 5.4 describes the approach I proposed, the new architecture, and training results. In chapter 6 details about the application's requirements, design, and testing are illustrated. Second to last, chapter 7 will present another use-case I have encountered in my recent research work, and finally, in chapter 8 I will conclude and wrap up with future work.

Chapter 2

Problem definition - Eye-Tracking

The problem is defined as building a system that can take as input an image of a person (taken using the laptop's built-in camera) and will output the 3D gaze vector of that person. A diagram of the use-case I am referring to is shown in Figure 2.1. This system should have low latency such that it can be part of a real-time eye-tracking tool. One such cycle is represented in Figure 2.2.



(a) Laptop setup (b) Feed of the built-in camera (c) 3D gaze prediction

Figure 2.1: Use-case: Eye-Tracking as a Human-Computer Interface

As one can observe from Figure 2.3 this problem is based on identifying some key patterns (e.g. where is the iris located inside the sclera, in which direction is the face oriented etc.). It is easy for a human to roughly guess the direction in which the subject is looking, however it is almost impossible to program a computer to do this in an algorithmic way. My suggestion is to apply supervised machine learning techniques to this problem. My approach is to develop an appearance-based gaze estimation neural network model which I will be training on already existing datasets from the literature.

2.1 Formal description

This problem can be written formally in the following way.

$$\text{let } f, g : D \subseteq \mathbb{R}^{N \times N \times 3} \rightarrow C \subseteq \mathbb{R}^3 \tag{2.1}$$

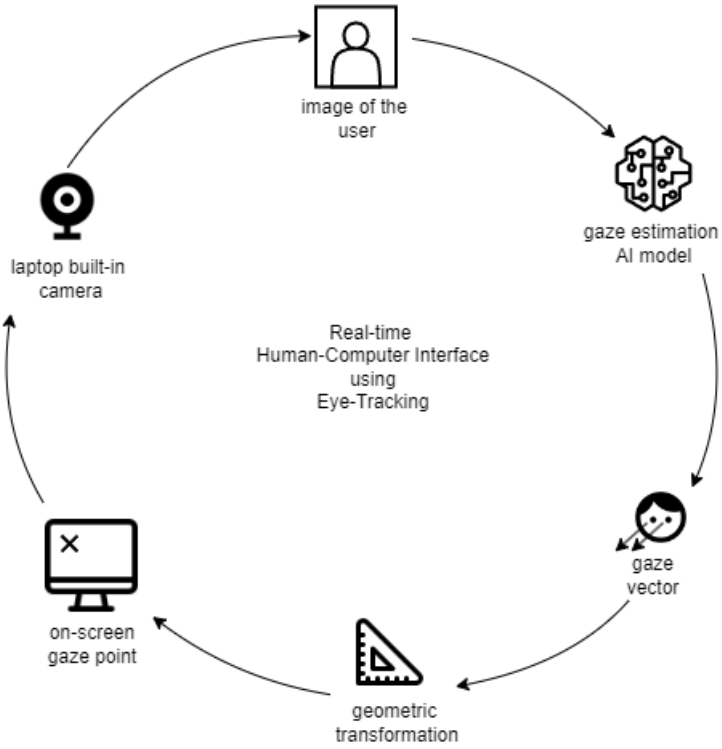
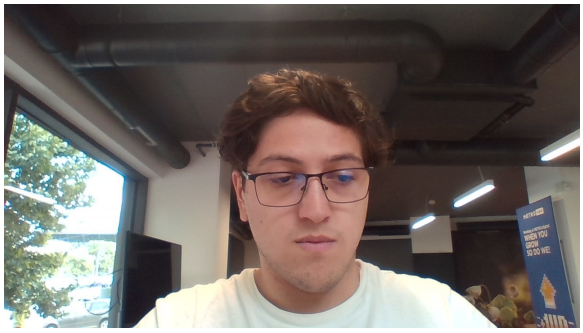
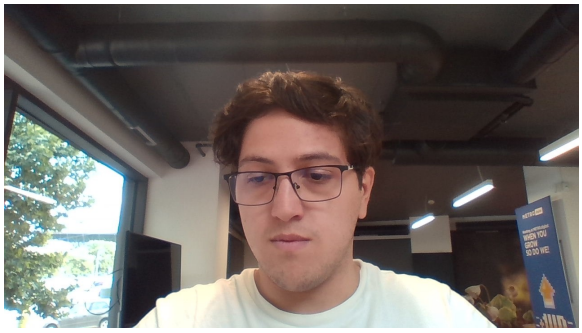


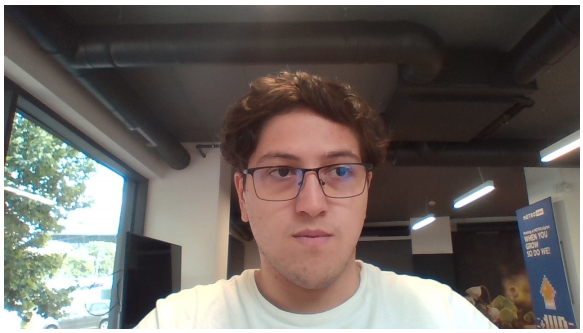
Figure 2.2: Cycle of a real-time interface based on the Eye-Tracking technology.



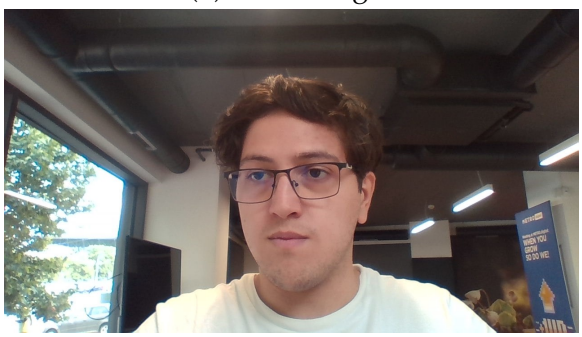
(a) Bottom left



(b) Bottom right



(c) Top left



(d) Top right

Figure 2.3: Subject looking towards each of the 4 corners of the screen. One can observe appearance-based patterns on the subject's face that indicate the direction of the gaze.

In other words, these functions should map from input x - a tensor of size NxNx3 representing a face image (where N is the image size), to an output y - a vector of size 3 representing the 3D gaze direction of that person.

I want to find the function f such that:

$$\min_{f} \|f(x) - \hat{y}\|$$

where $\hat{y}$ is the true gaze direction. This task is really hard to solve first-hand, so I approach the problem from another direction.

Assuming there are M unique pairs $(x, \hat{y})$ sampled from function f that form the set S, one can try to find a function g such that:

$$\min_{g} \sum_{(x,\hat{y})\in S} \|g(x) - \hat{y}\| \tag{2.2}$$

The function g will also be a rough approximation of function f. Its error decreases with the number of samples M.

$$g \approx f$$

If I assume that the samples cover a large portion of the input space D following the same data distribution as the input space, and M is large enough, then the function g will converge to f:

$$\lim_{M\to\infty} g = f$$

In the next practical chapters I will focus on finding the optimal function g that ensures Expression 2.2 and I will consider it a good approximation for the function f since the dataset I will introduce later is quite large.

2.2 Challenges

The toughest challenge is to find the optimal solution in this extensively large input space shown in Expression 2.1. In practice the solution should pay more attention to specific parts of the input images (e.g. eyes, face orientation) to the detriment of others (e.g. nose, mouth). Once a solution manages to achieve this, there will be more challenges in identifying very general patterns such that they can be applied to any subject. On the other hand, these patterns should also be sensitive enough to perform well on small eye movements.

Another challenge is represented by the constraints imposed by the system in which this solution will be integrated. The solution should achieve Real-time responsiveness. Real-time performance for interactive applications is usually consider to be between 10-100 ms response time. My goal is to achieve a 16.66 ms response

time (in other words 60 frames per second) in order to match the screen's refresh rate which one can argue is enough for daily use.

Two other challenge that may arise are regarding the dataset collection. One should identify a well suited dataset, large enough and with high variability (e.g. numerous subjects, good gaze angle coverage, diverse lighting conditions). Depending on the size of the dataset other challenge may occur: as resources (e.g. processing power, time) are limited, processing and training on a large dataset might be difficult.

Chapter 3

Background

3.1 Human-Computer Interface

Human-computer interaction is the field that studies different ways people communicate with computers. In fact, the first computers that appeared in the mid-19th century had no interactive human-computer interface (HCI) as the ones we are familiar with today, but programmers reprogrammed the computers using combinations of switches, cables, and plugboards. This trend was followed by punch cards used by IBM 80 series computers and later command-line interfaces. Nowadays, graphic user interfaces are most common, and they require additional hardware such as a monitor, a keyboard, or a mouse.

As shown on the Human-Computer interaction Wikipedia page [49] these HCIs can be categorized in:

1. sensor-based (e.g. mice and keyboards, joysticks, haptic sensors, touchscreen)
2. audio-based (e.g. speech recognition, noise detection)
3. visual-based (e.g. body movement, eye tracking, gesture recognition)

It is obvious that most of the commonly used HCIs are mainly sensor-based, which is the least natural interface humans can interact with. Moreover, most of these interfaces do not take into consideration accessibility for disabled people.

Recent studies are reevaluating the status quo in HCIs and are trying to come up with new, more natural interfaces while keeping an eye on accessibility.

For instance, Harada et al. [17] proposed a vocal joystick used for cursor control. This technology uses a multi-layer perceptron to classify between different vowel sounds mapped to a 2D space and then move the cursor accordingly. They also offer the possibility of performing clicks using the short consonant 'k'. This solution has a lower index of performance compared to mice; however, it has better performance than other speech-based control methods.

Another interesting piece of research is by Mohammadi et al. [32] where researchers developed an inductive tongue computer interface (See Figure 3.1) for people with severe disabilities to be able to control robotic devices such as prosthetic hands. The device has 18 inductive sensors that can be mapped to 18 different actions. The in-mouth hardware communicates wirelessly with the central unit. Experiments show that its effective throughput is 1.31 bps, compared to a gamepad joystick with a throughput equal to 3.22 bps.

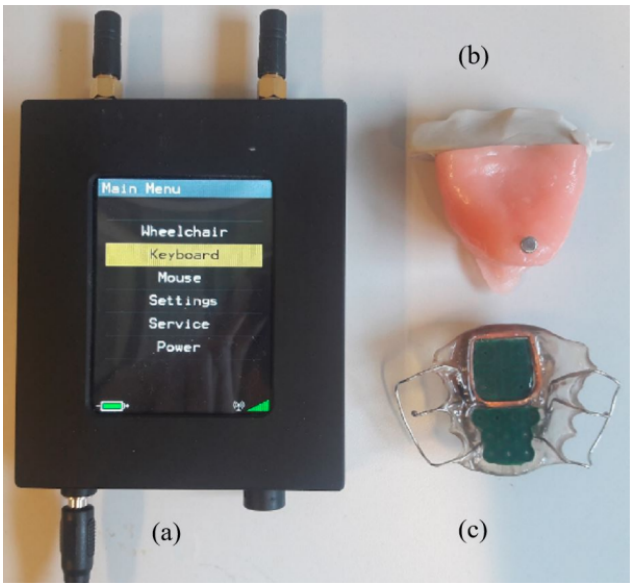


Figure 3.1: The inductive tongue control interface modules [32]. a: The central unit, b: The activation unit on a phantom tongue, c: The mouthpiece

A great deal of research effort has been put into analyzing the possibility of using feet as computer input. As shown in the work of Pakkanen et al. [37], feet are completely left aside in the process of computer interaction; however, conducted experiments show that foot controls are suitable for tasks that don't require high accuracy or fast execution time. The experiments consisted of controlling cursor movement with trackballs using both hands and feet.

The most exciting technology that is being researched at the moment is the brain-computer interface because of its limitless possibilities. The review by Nicolas-Alonso et al. [36] gathers all the research that has been going on between 1992 and 2012. The Different neuroimaging approaches that have been successful, such as EEG, fMRI, and NIRS. Nevertheless, more work needs to be done in order to assess the advantages and disadvantages (e.g., tissue damage, risk of infection) of different data acquisition methods. In the upcoming future, BCI is expected to seemlessly replace current HCI, but for now, the technological advances are yet to come.

The topic I want to explore in the next chapter is Eye-Tracking HCI because of its recent research advancements, accessibility, and non-intrusive applicability in day-

to-day life, while being one of the most natural interfaces out there.

3.2 Eye-Tracking

Eye-Tracking is the action of understanding, measuring, and making use of human eye movements. Before I dive into complex techniques applied successfully in eye-tracking, one must first know some basic concepts that stand as the basis for the research field. As explained by Klaib et al. [25] the human eye consists of different parts: retina, pupil, cornea, sclera, and iris. The identification and tracking of these parts are essential for developing eye-tracking technologies. There are two types of eye activities: saccades and fixations. A saccade is defined as fast movements between different gaze points, while a fixation is a longer period of time during which the subject is focusing on a specific object in his eye sight.

Klaib et al. [25] also classify the eye-tracking methods into different clusters: feature-based, appearance-based, and hybrids. Feature-based methods are dependent on indentifying parts of the human eye and don't work well in cases where these parts are hard to observe (such as poor lighting conditions or extreme head movements), while their counterpart, appearance-based solutions, make use of 2D images of the subject's face or eyes in order to map their gaze to a 2D imaginary plane. Solutions can also be structured as hardware or software products. For instance, some consumer-ready hardware devices used for eye tracking are Tobii Eye Tracker, iMotion, Varjo, and even newer devices such as Apple Vision Pro revolutionize the way people control computers. However, these products are expensive, and huge efforts are going into developing software-based solutions that predict eye gaze based on camera feed (such as a laptop camera). This goal is challenging in its own nature because of the amount of data required to create a solution that generalizes well to different cameras, subjects, lightning conditions, and gaze angles.

Some paths that have been explored in the research area and are presented by Klaib et al. [25] are the Scleral Search Coil Technique, infrared oculography (IOG), electrooculography (EOG), and video oculography (VOG). The former one uses contact lenses with wires connected to mirrors that reflect infrared light (See Figure 3.2). Based on the magnetic field, mirrors are moving so that eye trackers can measure the signal. Many laboratories tend to avoid this method because it is invasive and expensive; however, it has been shown to have promising results.

IOGs are based on infrared light that is emitted by a wearable device (such as light-emitting glasses). Light is reflected by the sclera, and the receiver measures the signal. Researchers from the Faculty of Electronics, Telecommunications and Information Technology, "Gheorghe Asachi" Technical University, Iaşi, have proposed a new technique for improving pupil detection [9]. An advantage of this method is

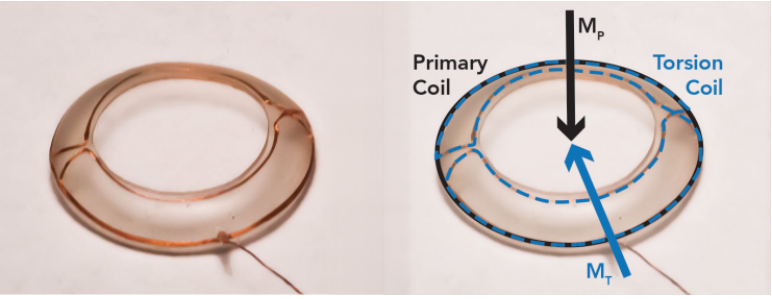


Figure 3.2: Scleral search coil contact lenses [48]

its high accuracy in identifying eye blinking. EOG is a technique based on sensors attached to the area around the eye that identify the electric field. This solution is not practical for everyday use but for laboratories or in the medical field.

Finally, video oculography (VOG) is the most explored method in recent years because of great advances in image processing and machine learning. VOG uses one or multiple cameras to record images of the subject's face or eyes. There are two types of VOG solutions: head-mounted or remote. Head-mounted devices do not account for head movements and are less practical because of their setup, while remote solutions are really practical in the case of mobile devices (phones, tablets), laptops, or any type of device with a front-facing camera.

Eye-Tracking studies are located at the intersection of Machine Learning, Internet of Things, and Cloud Computing as pointed out by Klaib et al. [25]. Machine Learning offers methods for a computer to learn patterns from previously recorded data and generalize them to new data. Some algorithms that have been applied successfully in eye-tracking are Neural Networks, Convolutional Neural Networks (CNN), Naive Bayes Support, Vector Machine (SVM), and Random Forest. Great improvements have been recorded using deep CNNs.

Internet of Things (IoT) offers unlimited possibilities for different devices to interact with one another through wireless networks building an inter-connected system that aims to improve the quality of life. Cloud computing comes in to solve the challenge of low processing power of local computers while model training as well as allowing fast access to large amounts of eye tracking data.

When it comes to applications of eye tracking technologies Klaib et al. [25] indentified 4 main fields:

1. Healthcare
2. HCI and accessibility
3. Education
4. Car assistant

In healthcare Keller et al. [22] have developed a system that helped patients suffering from amyotrophic lateral sclerosis (ALS) to navigate trough a predefined interface, Lozano et al. [30] developed a system for identifying individuals with autism spectrum disorder (ASD), while Garcia-Zapirain et al. [51] used an eye tracking system to improve attention skills for children with attention deficit hyperactivity disorder (ADHD).

Second of all, there have been attempts to develop systems to interact with computers via eye tracking (e.g. pointing, moving screen objects, selecting menus, dragging, and dropping). One common problem indentified in eye tracking HCIs is the 'Midas Touch' problem that occurs when selecting items on the screen is done by gazing at them. A solution was to show a confirmation pop-op on the screen examplified by Mohan et al. [33].

Other fields that benefit from eye tracking HCI solutions are AR/VR, gaming (reducing GPU load and generating high-quality content only for the gazed area - this technique is called "foveated rendering" in VR applications). Some research focused on helping people with hand disabilities who cannot use mouse and keyboard.

Thirdly, eye tracking is used for education purposes for assessing the level of understanding and measuring the interest and attention by Sun et al. [45]. Another important use case explored by Anusha et al.[6] is in monitoring drivers, and building a collision avoidance warning system by Park et al. [39]. Eye tracking technologies have been used in other scenarios such as analyzing consumer psychology though measuring online shopping interactions.

A more recent literature review by Ibragimov et al. [20] makes another comprehensive review of studies in the medical imagining field using Eye Tracking technologies. There are a lot more studies that use hardware eye trackers (e.g. Tobii 4C, ASH H6, EyeTribe) compare to the limited number of studies of software solution. Three main problems are studied in medical imagining:

1. reader performance
2. fatigue recognition from eye movements
3. gaze assistance for image annotation in order to build ML training databases

The work of Rustam Shadiev et al. [42] is a comprehensive review of immersive virtual reality (IVR) technologies that use eye-tracking methods. One of the main points is that Tobii and HTC Vive eye trackers are the most commonly used once. The sub-problems studied more frequent are: 1. educational technology (e.g. finding best suited materials for IVR, design virtual classrooms) and 2. Engineering and safety training (e.g. simulating different scenarios without unnecessary investments, reach a sufficient level of proficiency for a task with a high degree of danger).

3.3 Datasets

Before introducing existing Eye Tracking datasets in the literature, I must learn more about how researchers generalized the problem. First of all, the majority of datasets have two types of targets - 2D or 3D gaze estimations. The first category refers to 2D gaze positions on a plane (e.g. screen) while the other one refers to the gaze direction in a 3D world. Often datasets consist of full images of subjects, face images or only eye images. There have been multiple ways researchers have collected data, varying from using RGB cameras, depth cameras, infrared cameras to using webcams, and mobile devices (e.g. smartphones and laptops with front-facing cameras). Some of the most frequently used datasets for gaze estimation are shown in Table 3.1 alongside with their characteristics.

MPIIGaze dataset

Now I want to introduce MPIIGaze dataset collected by Zhang et al. [53]. The dataset was put together by recording participants outside the lab, in daily life scenarios, over several months. People were located in different locations, at different times, using different environment illumination, and having variable head poses. The experiments were recorder using laptop's built-in camera. Every 10 minutes a software notified the subject to look at a random sequence of 20 on-screen positions and press enter when they fixated their gaze. In total 213,659 images were taken from 15 participants. The dataset, has raw images of the subjects but they also provided eye images after normalisation.

MPIIFaceGaze subset provides processed face images that I am going to train my model on. Labels for eye landmarks, on-screen and 3D gaze targets, as well as estimated 3D head pose, and position of eyes in the camera coordinate system are provided for each image/subject. I chose the MPIIGaze dataset since it has exactly the same setup as my intended use-case (e.g. using laptop built-in camera and predicting 2D screen targets), and it has a good distribution of illumination, head poses, and gaze angles. It is also a medium size dataset, which allows me to perform several experiments.

Evaluation metrics

The evaluation metrics regularly used in literature for the eye-tracking task are:

1. Angular error
2. Euclidian distance (L2 norm)

The angular error is usually used for 3D gaze estimation where I want to compare the gaze direction groundtruth $\mathbf{g} \in \mathbb{R}^3$ to the prediction $\hat{\mathbf{g}} \in \mathbb{R}^3$. The formula for angular error is given by Equation 3.1. The Euclidian distance has been used for 2D gaze estimation. I denote groundtruth on-screen gaze position $\mathbf{p} \in \mathbb{R}^2$, and the prediction $\hat{\mathbf{p}} \in \mathbb{R}^2$, then the error can be computed using Equation 3.2.

$$L_{\text{angular}} = \arccos\left(\frac{\mathbf{g} \cdot \hat{\mathbf{g}}}{\|\mathbf{g}\|\|\hat{\mathbf{g}}\|}\right) \tag{3.1}$$

$$L_{\text{Euclidian}} = \|\mathbf{p} - \hat{\mathbf{p}}\| \tag{3.2}$$

Data pre-processing

Eye-Tracking shall not be done directly on raw images, since these often contain unnecessary information such as the background. Usually, scientists perform face or eye detection using facial landmarks and bounding boxes and then perform the training only on eye/face images. Performing unconstrained gaze estimation means taking into consideration illumination factors and different head poses. Next pre-processing phase is data rectification, which is used to remove subject rotation, translation and scaling, such that training can be done on normalized face/eye images. See Figure 3.3.

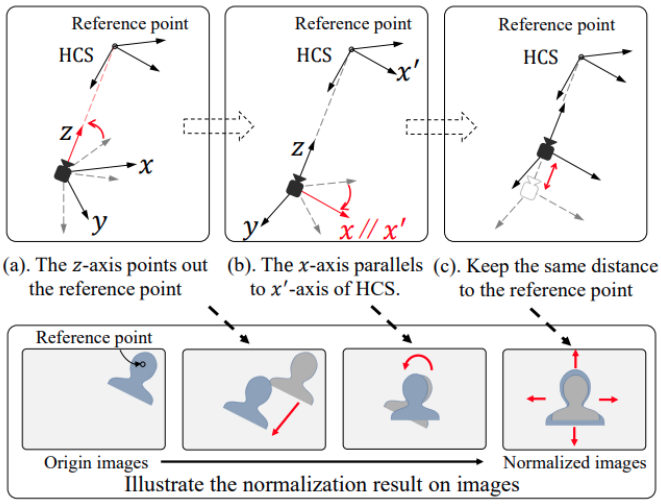


Figure 3.3: Data pre-processing for gaze estimation presented by Yihua Cheng et al. [10]

3.4 Related methods

In this sections I want to summaries machine learning methods that have been used for the gaze estimation task. Methods are presented in chronological order.

iTracker

Krafka et al. [27] introduced a new dataset called GazeCapture and proposed a new model called iTracker. They created an app that ran on iPhones and iPads. People were paid to solve some tracking tasks. By scaling this using crowdsourcing they manage to obtain a huge dataset of 2.5M frames. iTracker is a CNN model that takes as input images of left and right eyes, the face, and a face grid and outputs distances in cm from the camera on X and Y axis in the screen plane.

The architecture of the model is similar to the AlexNet proposed by Alex K. et al. [28]. Their architecture can be observer in Figure 3.4. Left and right eye CNN pipelines are sharing the same weights. They are using 4 CNN layers and 3 fully connected layers to get the 2 output values. They applied dark knowledge to reduce model complexity and achieved 0.05 sec inference time on an iPhone 6s. The evaluation metric used was euclidian distance. They achieved 1.71 and 2.53 cm error on phones and tablets respectively using this model fine-tuned for each device and orientation. Now they have done an ablation study and found out that all inputs contribute to the performance of the model. They also provided a calibration method using 13 fixed locations which resulted in improved performance of 1.34 and 2.12 cm error.

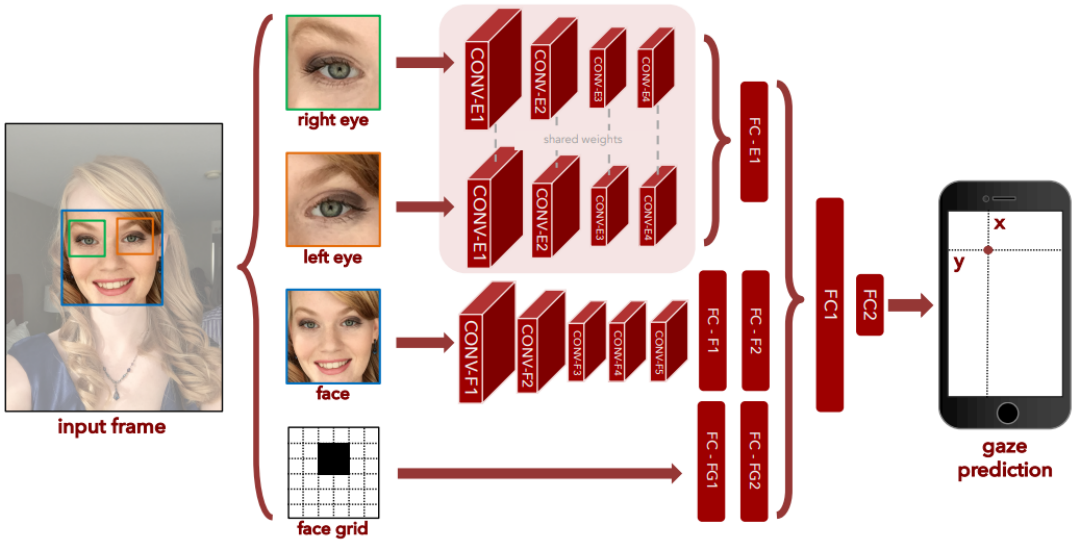


Figure 3.4: iTracker architecture [27]

GazeNet

GazeNet was initially proposed by Zhang et al. [53] together with the MPIIGaze dataset. See the dataset description in Table 3.1. This model takes as input an eye

image and the head angle vector and outputs a single 2D gaze vector (yaw/pitch). GazeNet is a multimodal CNN based on the LeNet architecture proposed by Yann LeCun et al. [29], having 2 pairs of convolution and max pooling layers followd by a fully connected layer. They use L2 loss to train their model. The model overview can be seen in Figure 3.5.

They compare this model with Random Forests, kNN, Adaptive Linear Regression, Support Vector Regression in 2 ways: cross-dataset evaluation and within-dataset evaluation. In the latter one, GazeNet outperforms the other models reaching a 6.3° mean angular error.

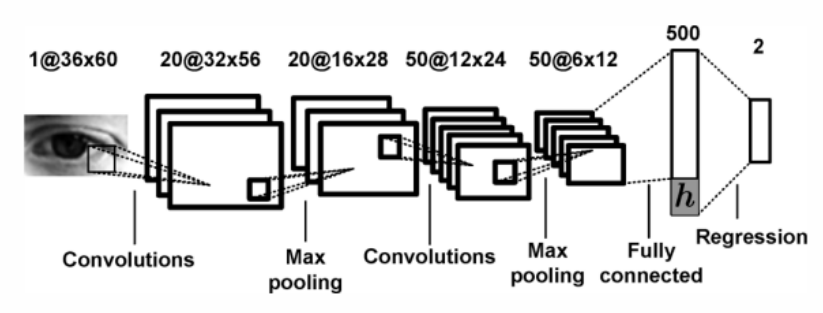


Figure 3.5: GazeNet architecture [53]

It's Written All Over Your Face

The same authors of the previously described model in Section 3.4 also proposed a new Spatial Weights CNN model [54]. This model takes as input only face images and outputs both 2D gaze on-screen location and 3D gaze vectors. The architecture can be observed in Figure 3.6. They used input data normalization introduced by Sugano et al. [44] and explained previously in Section 3.3. The backbone of the architecture is AlexNet, pretrained on LSVRC-2010 ImageNet. The spatial weights mechanism goal is to scale each feature in the feature map with a different weight, such that specific regions receive more attention than others (e.g. eyes).

They used L1 loss for training and used 5-fold cross-validation for evaluation. They achieved 4.2 cm error on 2D gaze estimation and 4.8 °angular error on 3D gaze estimation, exceeding state-of-the-art performance.

Gaze360

Petr Kellnhofer et al. [23] introduced a new video-based dataset called Gaze360 and proposed a model with the same name that exploits the temporal continuity of the gaze. Their model consists of ImageNet-pretrained ResNet-18 backbone proposed

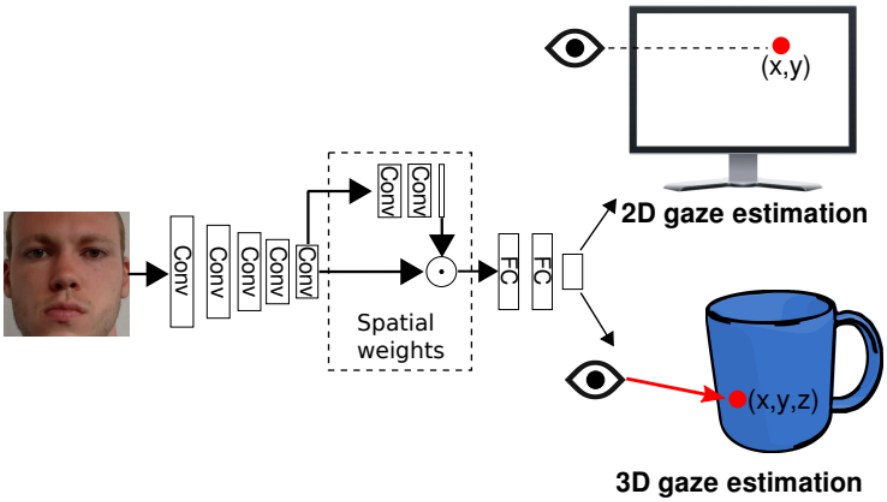


Figure 3.6: Spatial Weights CNN architecture [54]

by Kaiming He et al. [18] and a bidirectional Long Short-Term Memory initially proposed by Alex Graves et al. [15] (Bi-LSTM). Input is a sequence of 7 face images and the output is a single 3D gaze prediction and an error estimation term. They obtain an 11.1° angular error for the front facing subset of Gaze360 and 9.9° by training on Gaze360 and testing on MPIIFaceGaze. The model can be observed in Figure 3.7.

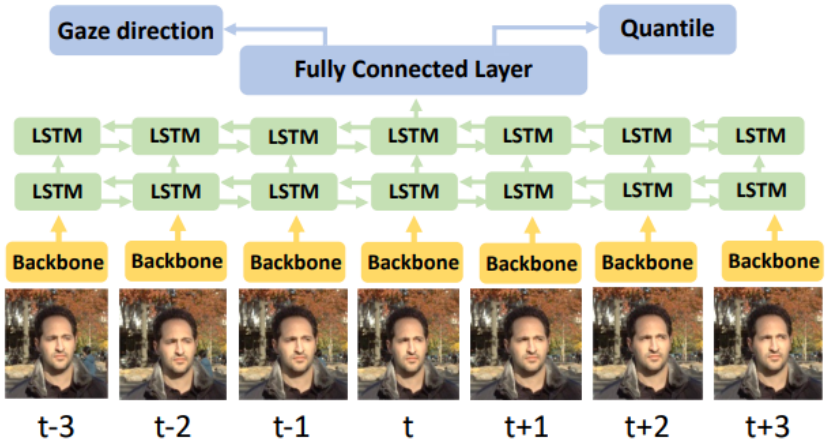


Figure 3.7: Gaze360 architecture [23]

MCGaze

The name of the model presented by Guan Yiran et al. [16] comes from Multi-Clue gaze estimation, and it is based on capturing spatial-temporal interaction context

among 3 features: head, face, and eye. Input of their model is a video clip, and the output is the 3D gaze estimation for each frame. They used a ResNet-50-FPN pretrained on ImageNet-1K as the backbone and the transformer architecture to identify strong interactions among spatial and temporal dimensions. See Figure 3.8.

They use different losses for optimizing the clue localization heads and added temporal regularization terms for better temporal modeling. For evaluation they used angular error defined by Equation 3.1. Results show an angular error of 10.02° on the detectable faces of the Gaze360 dataset, exceeding state-of-the-art. They also performed an ablation study to demonstrate that each component boosts performance of the overall model.

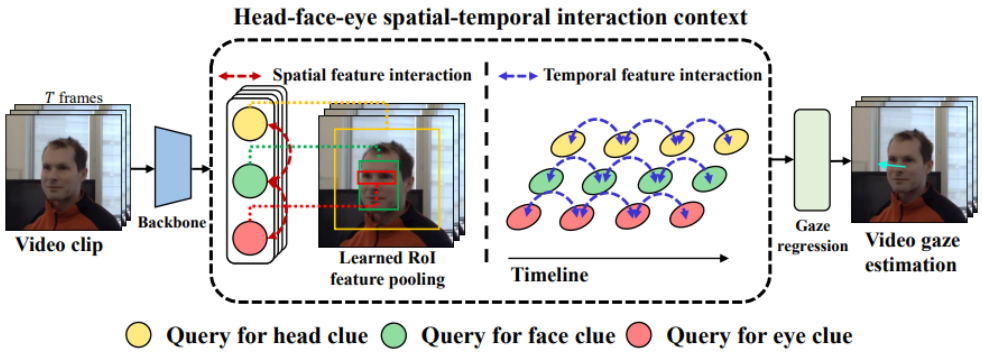


Figure 3.8: MCGaze architecture [16]

FaZE

The name of the model introduced by Seonwook Park et al. [40] comes from Few-shot Adaptive GaZE Estimation, and the idea behind it is to propose model that can be calibrated for each person specifically. The model is using the MAML meta-learning algorithm and an Encoder-Decoder architecture they named Disentangling Transforming (DT-ED). Their input is an image and output is a 3D gaze vector. They used arccos loss for training. They obtain an angular error of 3.18° on GazeCapture and 3.42° on MPIIGaze with as low as 3 calibration samples which improves the state of the art.

L2CS-Net

Ahmed et al. [4] proposed a CNN-based model that makes use of a pretrained ResNet-50 backbone and different regression branches for each gaze angle. See Figure 3.9. Inputs are face images and output are yaw and pitch gaze angles. As the loss function they used a combination of cross entropy loss (for classifying the

binned prediction) and the mean squared error between the groundtruth and the expected value of the probability distribution. They trained on Gaze360 and MPIIGaze datasets. They used the data pre-processing described earlier in Section 3.3. They obtained 3.92° angular error on MPIIFaceGaze and 9.02° on the front facing subset of Gaze360.

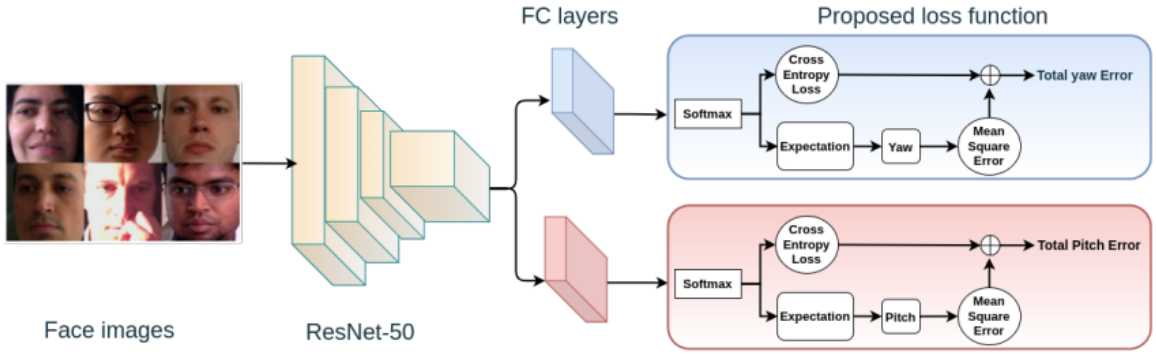


Figure 3.9: L2CS-Net architecture [4]

Dataset	Size	Description	Models trained on it
EyeDiap [34] 2014 Idiap Research Institute	16 subjects 94 videos 1920x1080 25 fps	collected in laboratory full face images 2D and 3D targets	**RecurrentGaze** [38] (3.4 angular error)
MPIIGaze [55] 2015 Max Plank Institute **MPIIFaceGaze** [54] subset 2017 Max Plank Institute	MPIIGaze: 15 subjects 213k images 2 GB MPIIFaceGaze: 15 subjects 45k images 900 MB	collected using laptops in daily life 2D and 3D targets MPIIGaze: eye images MPIIFaceGaze: face images	**Faze** [40] (3.1*) **L2CS** [4] (3.9*) **RT-GENE** [12] (4.3*) **GazeNet** [53] (6.3*) *angular error
GazeCapture [27] 2016 University of Georgia MIT Max Planck Institute	1474 subjects 2.4M images 136 GB	collected using mobile devices (tablets, smartphones) in daily life full faces 2D targets	**iTracker** [27] (1.34* on smartphones and 2.12* on tablets) **EnsembleCalibration** [7] (1.77*) **TinyTracker** [8] (2.34*) *L2 loss in cm
Gaze360 [23] 2019 MIT Toyota Research Institute	238 subjects 172k images 28 GB	collected indoor/outdoor wide range of head positions greater distances between camera and subject full faces 3D targets	**Gaze360** [23] (13.5*) **Weakly-supervised ResNet-18** [26] (12.94*) **L2CS** [4] (10.41*) **MCGaze** [16] (10.02*) *angular error
ETH-XGaze [52] 2020 ETH Zurich Google	110 subjects 1.1M images high resolution images $>$ 130 GB	collected in laboratory extreme head poses full face images 2D and 3D targets	**ETH-XGaze ResNet** [52] (4.7 angular error)

Table 3.1: Most used datasets in eye tracking literature

Chapter 4

Deep Learning

4.1 Introduction to Deep Learning

Software engineering is a broad field that encapsulates multiple other subfields (Artificial intelligence, web development, game development, security, embedded systems etc.). Artificial intelligence can also be splitted into other smaller fields such as evolutionary algorithms, machine learning, robotics, intelligent agents. I am interested in machine learning because it studies algorithms that use data to imitate the way humans learn, gradually improving performance without being explicitly programmed to do a specific task. Some of the machine learning subfields are reinforcement learning, deep learning, unsupervised learning. Finally I want to specialize on Deep Learning (See Figure 4.1) which is a specific type of machine learning that uses deep architectures to learn far more complex patterns.

4.2 Neural Networks architectures in Computer Vision

The basic architecture that stands as the basis of machine learning is the Neural Network (NN). This structure is conceived to mimmic how the human brain works. In fact NN nodes are called neurons and edges between them are similar to dendrites and axons. The goal of this structure is to learn patterns from input data and be able to map these inputs to output data. Each edge in a NN has a weight that multiplies with the neuron's output to obtain new values. By forward propagating the input data through the network a function of the input data is computed. This function is an approximation of the mapping function from input X to output Y. A simple Neural Network architecture with 3 layers of sizes 3x4x4x1 can be seen in figure 4.2.

The key property of a NN is the ability to improve by training on labeled data. This is done by back-propagating the derivative of the error with respect to each weight and updating it accordingly. Neural Networks can sinthesize unique rela-

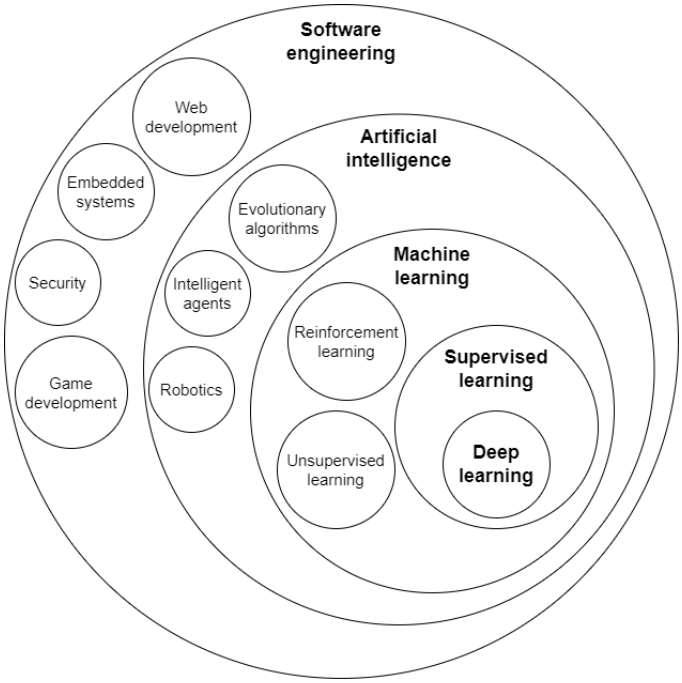


Figure 4.1: Hierarchy of fields that englobe Deep Learning in the broader field of Software Engineering

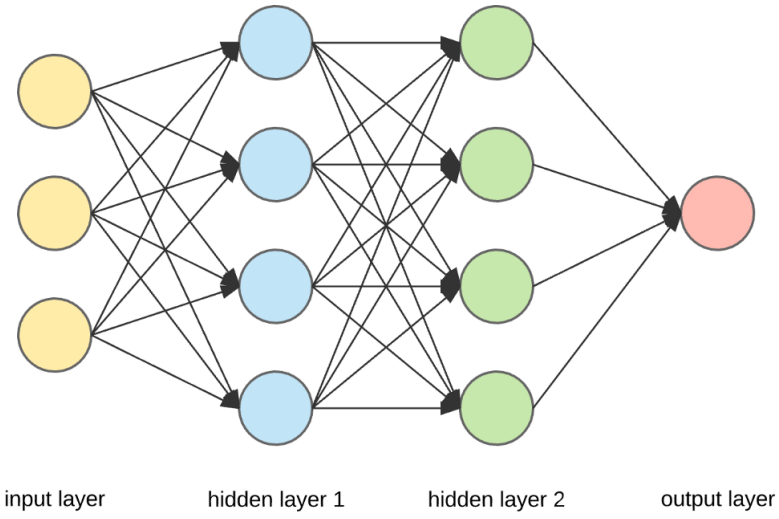


Figure 4.2: Neural network archtecture [14]

tionships between each 2 input data points, however this is not always the expected behaviour. For instance, in an image one does not differentiate if a face is two pixels to the right. It is still the same feature and should be learned by the same weights. That is the motivation of how Convolutional Neural Networks (CNNs) were initally proposed by Yann LeCun et al. [29]. This architecture reuses weights to indentify patterns in each different location in the input image. By doing this, the number of unique weights that have to be trained is lowered significantly.

Usually NN had a limited amount of layers one on top of each other. When CNN was first announced, this limitation dissapeared and scientists have built deeper architectures which can identify more complex patterns in an image. This architectures are generally reffered to as Deep Neural Networks, while the subfield is called Deep Learning. CNN are called this way since they are made of convolutional layers. These layers have weights that are structured in small K by K matrices that are applied/convoluted over each position in the input matrix. Over the years scientists have come up with better and better architectural designs for CNNs.

Everything started with the LeNet model proposed by Yann LeCun et al. [29] in 1998, and its architecture can be seen in Figure 4.3. He introduces convolutional layers as well as pooling layers, but used filters of varying size. Pooling layers reduce the size of the input image without using weights. This model was initally trained on MNIST dataset. Next AlexNet developed by Alex Krizhevsky et al. [28] was published in 2012, its architecture can be seen in Figure 4.4. It introduces max pooling (taking the maximum of an k x k area) as well as padding and stride for convolutional operations. Padding came as a solution to the fact that pixels at the border of the image influenced a smaller number of feature maps. Stride means that convolutional operations were not applied to each position possible in the input matrix, but jumping over by n squares, horizontally and vertically.

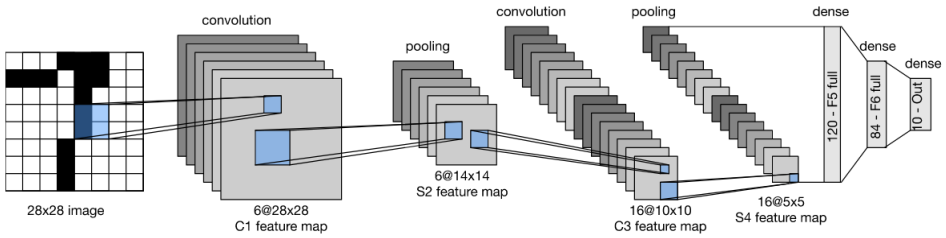


Figure 4.3: LeNet architecture [29]

VGG (2014) introduced by Karen Simonyan et al. [43] was a major jump ahead. They demonstrated that kernels of size larger than 3 (e.g. 5, 7) can be reproduced by using two or more layers of convolution with size 3. That means that a regular 5x5 kernel with 25 weights can be restructured using 2 3x3 kernels with a total of 18 weights. VGG-19 architecture can be seen in Figure 4.5. GoogLeNet (2014) proposed

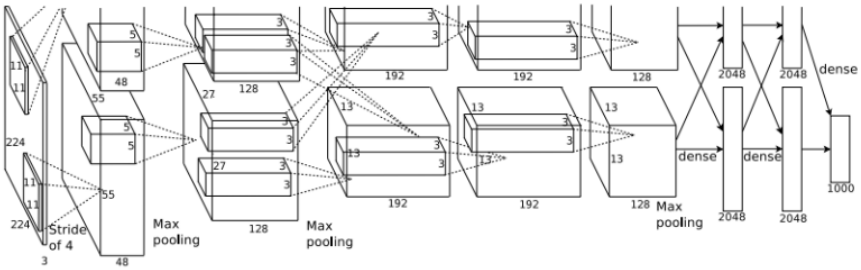


Figure 4.4: AlexNet architecture [28]

by Christian Szegedy et al. [46] first introduced inception modules. Their idea was to use multiple filter sizes within the same layer and then concatenate their results. This allows the network to capture features at multiple scales efficiently. They were also first to use 1x1 convolutions to reduce channel size (layer similar to a fully connected layer in a NN). On top of that, they had to deal with the vanishing gradients problem. To combat this, they have used intermediate classifier heads, such that the final loss is a combination of the intermediate losses and final loss. The GoogLeNet architecture is shown in Figure 4.6.

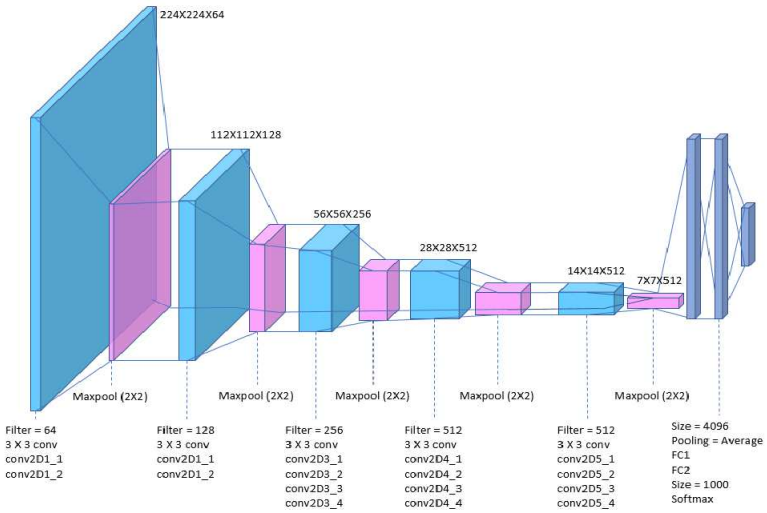


Figure 4.5: VGG-19 architecture [43]

A huge breakthrough was ResNet (2015) introduced by Kaiming He et al. [18]. They solved the problem of vanishing gradient problem which state that deeper architecture tend to be harder to train bc the backpropagation over a lot of layers minimizes the error. They achieved this by using residual blocks. These residual blocks had two consecutive convolutional layers, but also a skip connections that was added back to the ouput of the normal branch. This skip connection allowed the error to backpropagate without loss through the network. Their architecture can be observer in Figure 4.7 They were first to also use Batch Normalization. This architecture is very popular, and usually people reuse a pretrained ResNet model

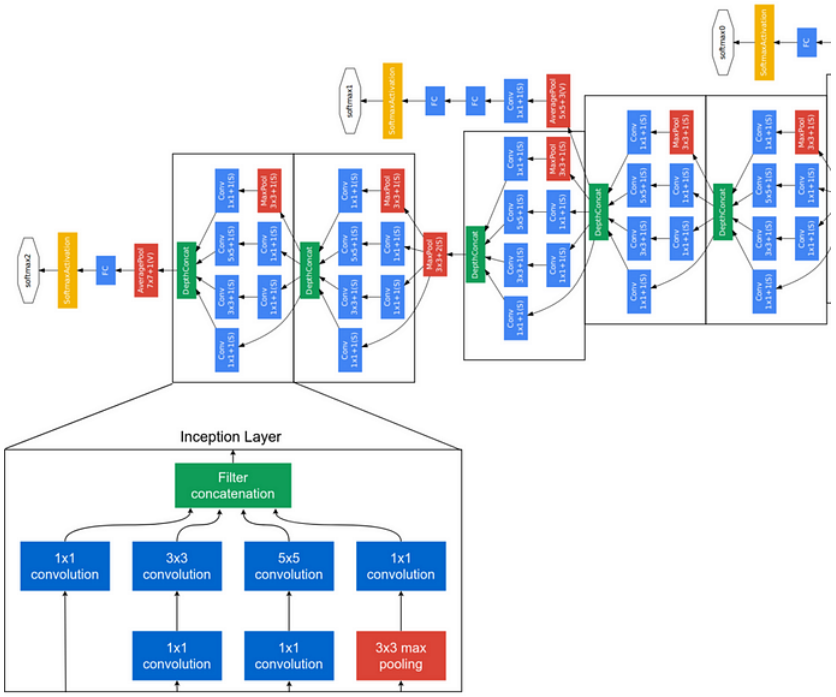


Figure 4.6: GoogLeNet architecture [46]

(of varying size 18 through 50) to indentify basic features in an image.

Other architectures that should be mentioned are ResNext proposed by Saining Xie et al. [50] which uses multiple branches in a single residual block, and MobileNets introduced by Andrew G. Howard et al. [19] (2017) that have come up with an efficient way of separating intensive computational convolutional operations into three phases: separating channels, performing convolution of each channel with a filter and then performing 1x1 convolution (pointwise convolution) to get the same receptive field. The method is called depth-wise separable convolutions. The architecture is called MobileNets since they can also be run on devices with low computational power. The MobileNet diagram can be seen in Figure 4.8

4.3 Key Concepts

In the next section I will define some key concepts I used in the upcoming chapters.

Batch size and epochs

A batch is a set of samples that are simultaneously fed through the network. After a batch of training data is done processing, the back-propagation algorithm is executed for all samples in the batch and the update is only done once per batch. Usually people split the training data into mini-batches (e.g. of size 16, 32, 64) such that they benefit from parallel execution and also get to update the weights more

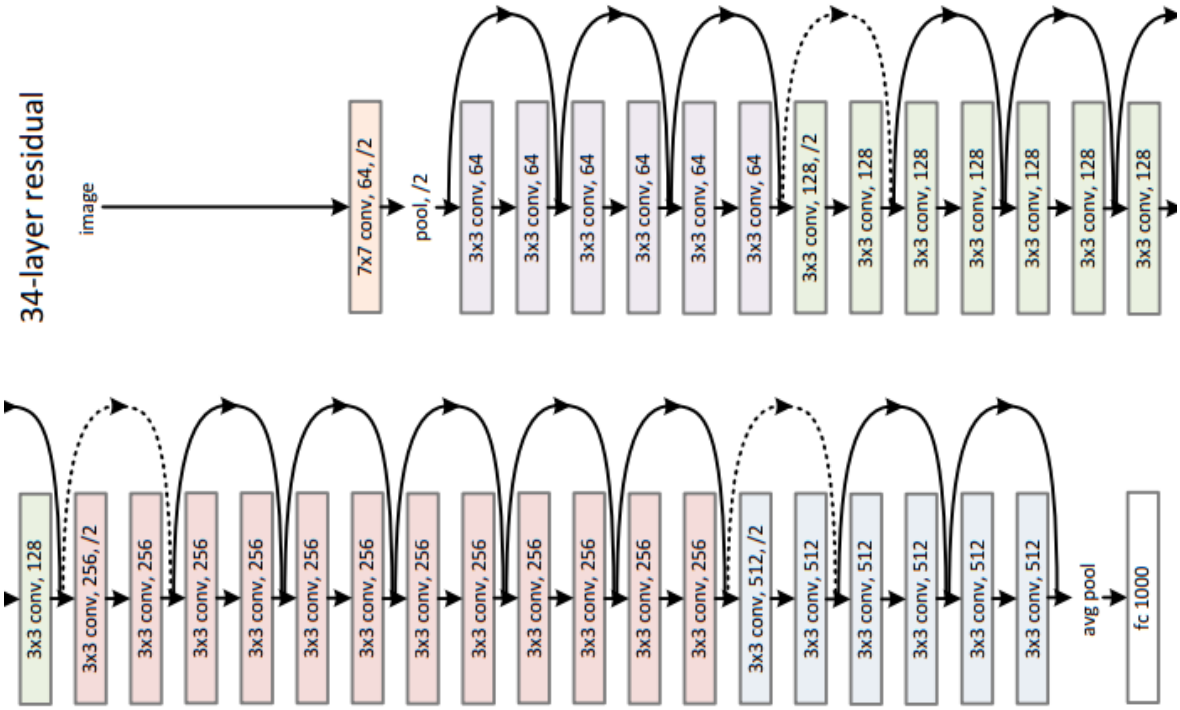


Figure 4.7: ResNet-34 architecture [18]

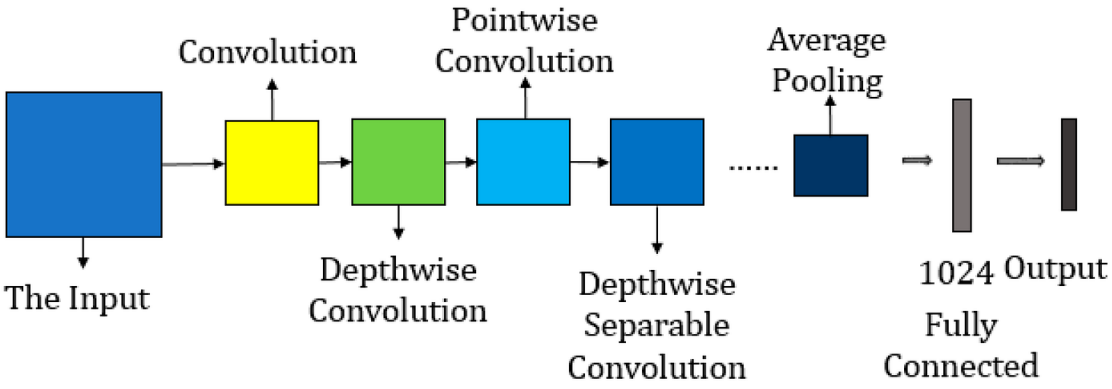


Figure 4.8: MobileNet architecture [35]

frequently.

An epoch is a unit of training that consists of all training samples to be fed once to the model. Usually models are trained for 10-50 epochs such that the model gets to see all samples multiple times and manages to lower the loss after every epoch. Some people do not explicitly define the number of training epochs, but they stop the training if for the last k epochs the loss did not improve. In such cases it is a good practice to save the model with the lowest loss not necessarily the last model.

K-Fold cross validation

Let k be a constant factor. One can split any dataset randomly in k subsets. K-Fold cross validation is a method of splitting these k subsets between training and validation. This method involves training k models, each on k-1 subsets and then testing each one on the remaning subset.

Backbone

The backbone is the part of neural network that extracts features from the input data and outputs a feature-map that can be further processed. Usually, backbones consist of recognized architectures that were validated beforhand. Sometimes, the backbone can be pretrained and it's weights can be freezed or trained alongside with the whole network.

Special layers

Adaptive 2D pooling [13] layer's scope is to downsample to the preferred 2D dimension, by splitting the matrix into different equal-size regions and then apply an aggregation function on values in each region. For instance, adaptive 2d average pooling layer takes the average of each region, such that it outputs the given dimension.

Batch normalization proposed by Sergey Ioffe et al. [21] is used to normalize a feature-map at a point in a neural network. It has two trainable parameters that represent the mean and variance of all values in a channel. When in evaluation mode, it automatically uses the learned parameters to normalize new feature-maps.

Activation functions

An activation function is a function applied to a feature-map to delinearize it, such that the model can learn patterns that are not linear in nature. One such activation

function is ReLU - Rectified Linear Unit proposed by Abien Fred Agarap [5], which sets a lower bound of zero to each value in the feature-map.

Softmax activation function introduced by Rumelhart et al. [41] is also often used, since it transforms a vector of values with no strict criterion into probabilities (values between 0 and 1 that sum up to 1).

Loss functions

A loss function is used to quantify the performance of a model by comparing prediction to the groundtruth. The Mean Squared Error (MSE) is often use in regression problems where the squared error is significant for the task (e.g. for predicting prices of stocks).

Cross-Entropy Loss, initially presented by Rumelhart et al. [41] is mainly used for classification tasks (e.g. image of a cat or a dog) and it measures the performance of a model whose outputs are probabilities between 0 and 1.

Learning rate and the optimizer

The learning rate is a training hyperparameter that scales the step taken when optimizing weights. A large learning rate (e.g. 0.1, 0.01) will prevent the model from reaching deep local minima. On the other hand a small learning rate (0.00001, 0.000001) means a very slow convergence, and consequently longer training sessions.

The optimizer is a method to update the weights of a neural network. There are multiple versions of optimizers that work best in different situations.The Adaptive Moment Estimation (Adam) optimzer proposed by Diederik P. Kingma et al. [24], and which combines momentum with adaptive learning rate is predominantly used.

Underfitting and overfitting

When training a model on a specific task, there two extreme situation that might arise. Not being able to learn and understand the data, which can be observed by observing a high training loss. This situation is often reffered to as underfitting.

On the other hand, the model might learn a perfect representation of the training data (small training loss) including outliers and perform much worse on the test data (high test loss). This situation is called overfitting.

Chapter 5

Proposed approach

5.1 Model development pipeline

Overview

To begin this chapter with, I want to make a brief description of the procedure I am using in the next experiments including data preparation, training, and inferencing machine learning models. The procedure is represented visually in Figure 5.1. The dataset pre-processing part is done only once at the beginning, while the training loop is ran for each new experiment separately.

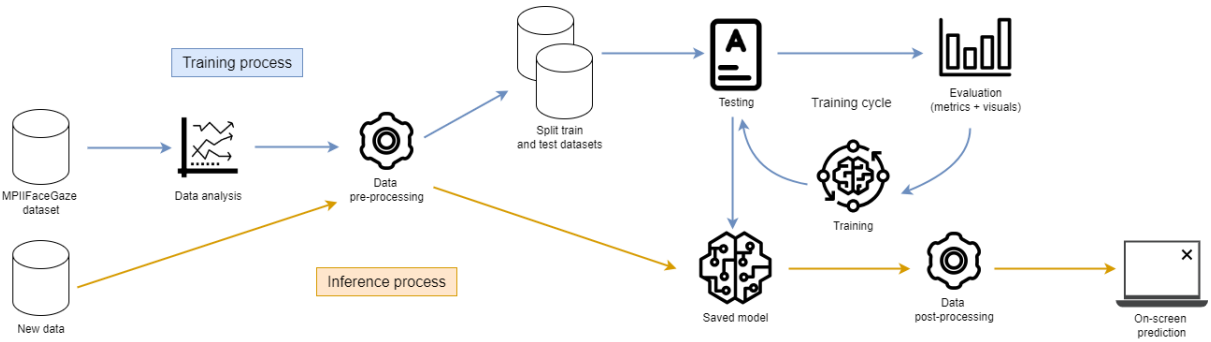


Figure 5.1: Processing procedure preview

Training pipeline design

In designing the training pipeline I opted for an experiment driven setup. In short, the training pipeline is the same for all models, however there is a configuration file that should be adapted for each experiment individually. The pipeline can be observed in Figure 5.2. The pipeline consists of a training script with multiple train-test split strategies, the actual training loop, and the evaluation procedure. The

training pipeline is implemented using the PyTorch frameworks. Experiments are set to be deterministic and reproduceable.

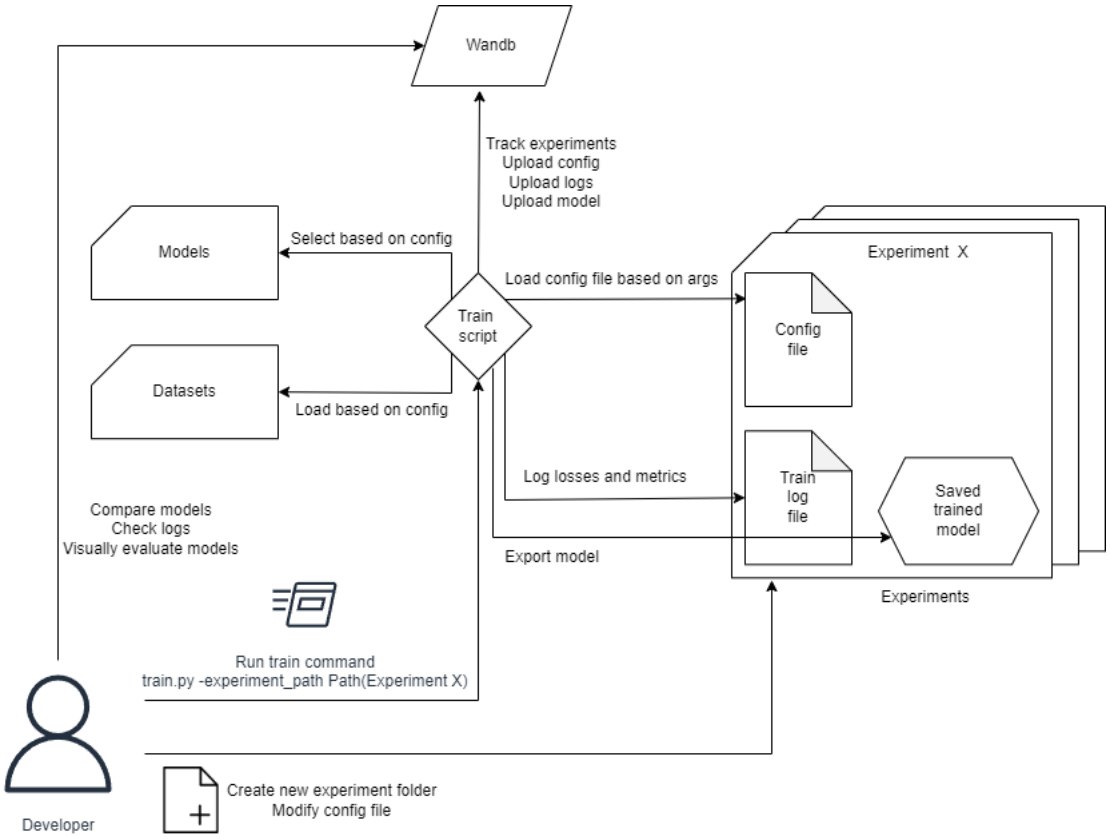


Figure 5.2: Training pipeline architecture

Configurations

I provide a configuration template that allows the user to modify the folding strategy, model backbone, number of epochs, batch size, learning rate. The option to change the seed is also provided in order to change the randomness seed. I have also added a pipeline-testing option which trains the model on a limited number of samples and epochs such that the pipeline can be validated faster. This configuration file is to be copied over for every new experiment and defines the parameter of the model, the dataset, and the training script. One example of a configuration file can be seen in Figure 5.3

Logging and tracking

The setup has been enhanced with logging at each step and experiment tracking using the Wandb web interface [2]. A preview of the Wandb interface is shown in Figure 5.4. Logging happens in real time in the command line interface as well as

```
{
    "data": {
        "folding_strategy": 1,
        "dataset_name": "MPIIFaceGaze",
        "dataset_dir": "/mnt/d/Downloads/MPIIFaceGaze/ProcessedByMe"
    },
    "model": {
        "backbone": "ResNet18",
        "bins": 28
    },
    "train": {
        "num_epochs": 5,
        "batch_size": 24,
        "learning_rate": 0.0001,
        "optimizer": "Adam",
        "use_gpu": true,
        "seed": 2,
        "is_pipeline_test": false
    }
}
```

Figure 5.3: Example of a configuration file

a training log file. An example training log file is shown in Figure 5.5. I also log information about experiment configuration, training/evaluation losses over time, and the evaluation metric at different timestamps of the training process. On top of that, I provide a visual way to assess the improvement of the model over time by logging 12 sample images with labels and prediction gaze vectors each epoch.

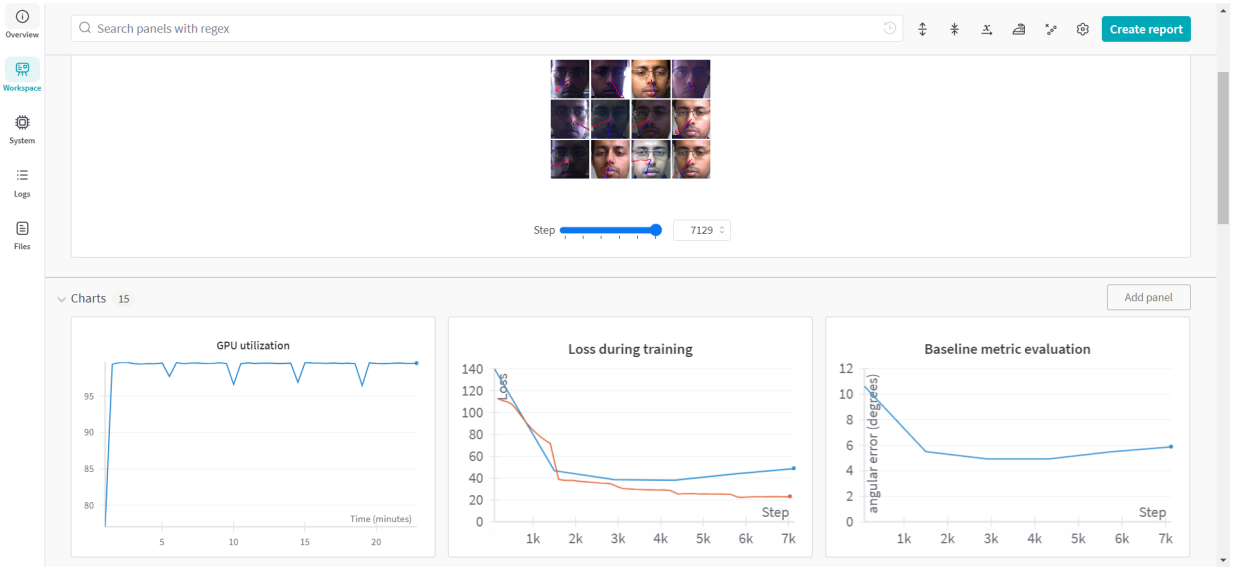


Figure 5.4: Preview of Wandb interface

5.2 Data analysis

I am going to train my model on the MPIIFaceGaze dataset. In the next sections I want to describe how the data is structured, what labels am I using, how data is distributed, and what data preparation techniques I use in my approach.

```
01/05/2024 23:54:01 [INFO] -------- Starting evaluation on test --------
01/05/2024 23:54:31 [INFO] -------- Epoch: [0/5], Batch: [100/125], Test loss: 164.3934, Mae: 11.6558 --------
01/05/2024 23:54:39 [INFO] -------- Epoch: [0/5], Batch: [125/125], Test loss: 164.6211, Mae: 11.6583 --------
01/05/2024 23:54:41 [INFO] -------- Starting training --------
01/05/2024 23:56:17 [INFO] -------- Epoch: [1/5], Batch: [100/1750], Train loss: 112.9618, Train pitch loss: 79.7050, Train yaw loss: 33.2568
01/05/2024 23:57:53 [INFO] -------- Epoch: [1/5], Batch: [200/1750], Train loss: 111.2217, Train pitch loss: 79.3547, Train yaw loss: 31.8670
01/05/2024 23:59:30 [INFO] -------- Epoch: [1/5], Batch: [300/1750], Train loss: 109.4003, Train pitch loss: 78.2756, Train yaw loss: 31.1247
02/05/2024 00:01:06 [INFO] -------- Epoch: [1/5], Batch: [400/1750], Train loss: 107.4738, Train pitch loss: 77.0388, Train yaw loss: 30.4351
02/05/2024 00:02:42 [INFO] -------- Epoch: [1/5], Batch: [500/1750], Train loss: 104.7257, Train pitch loss: 74.8664, Train yaw loss: 29.8593
02/05/2024 00:04:18 [INFO] -------- Epoch: [1/5], Batch: [600/1750], Train loss: 100.8150, Train pitch loss: 71.6899, Train yaw loss: 29.1251
02/05/2024 00:05:54 [INFO] -------- Epoch: [1/5], Batch: [700/1750], Train loss: 96.5216, Train pitch loss: 68.0307, Train yaw loss: 28.4909
02/05/2024 00:07:29 [INFO] -------- Epoch: [1/5], Batch: [800/1750], Train loss: 92.1387, Train pitch loss: 64.4831, Train yaw loss: 27.6556
02/05/2024 00:09:05 [INFO] -------- Epoch: [1/5], Batch: [900/1750], Train loss: 88.1417, Train pitch loss: 61.2985, Train yaw loss: 26.8432
02/05/2024 00:10:41 [INFO] -------- Epoch: [1/5], Batch: [1000/1750], Train loss: 84.7138, Train pitch loss: 58.4612, Train yaw loss: 26.2526
02/05/2024 00:12:17 [INFO] -------- Epoch: [1/5], Batch: [1100/1750], Train loss: 81.5616, Train pitch loss: 55.9683, Train yaw loss: 25.5933
02/05/2024 00:13:53 [INFO] -------- Epoch: [1/5], Batch: [1200/1750], Train loss: 78.7459, Train pitch loss: 53.7141, Train yaw loss: 25.0319
02/05/2024 00:15:29 [INFO] -------- Epoch: [1/5], Batch: [1300/1750], Train loss: 76.3143, Train pitch loss: 51.7964, Train yaw loss: 24.5179
02/05/2024 00:17:05 [INFO] -------- Epoch: [1/5], Batch: [1400/1750], Train loss: 73.9470, Train pitch loss: 49.9706, Train yaw loss: 23.9764
02/05/2024 00:18:40 [INFO] -------- Epoch: [1/5], Batch: [1500/1750], Train loss: 71.8599, Train pitch loss: 48.3911, Train yaw loss: 23.4688
02/05/2024 00:20:16 [INFO] -------- Epoch: [1/5], Batch: [1600/1750], Train loss: 70.1043, Train pitch loss: 47.0930, Train yaw loss: 23.0112
02/05/2024 00:21:52 [INFO] -------- Epoch: [1/5], Batch: [1700/1750], Train loss: 68.4525, Train pitch loss: 45.8455, Train yaw loss: 22.6069
02/05/2024 00:22:40 [INFO] -------- Epoch: [1/5], Batch: [1750/1750], Train loss: 67.7030, Train pitch loss: 45.2654, Train yaw loss: 22.4376
02/05/2024 00:22:40 [INFO] -------- Starting evaluation on test --------
02/05/2024 00:23:10 [INFO] -------- Epoch: [1/5], Batch: [100/125], Test loss: 44.0357, Mae: 5.4273 --------
```

Figure 5.5: Training log file example

Dataset structure

The original MPIIFaceGaze dataset is structured as follows. There are 15 folders each containing data for a participant. Each participant has a calibration folder which contains camera parameters, the monitor pose in the camera coordinate system, and monitor screen size in mm and pixels. Actual images are stored in different folders for each day of the data collecting process. Images of size 1280x720 are RGB and have a black background, allowing only the face to be seen. There are about 100 images in each folder and variable number of days for each person. Labels come as a csv file with 28 dimensions, where each row corresponds to an image:

Dimension 1 image path

Dimension 2-3 gaze screen coordinates (in pixels)

Dimension 4-15 locations of six facial landmarks (4 eye corners and 2 mouth corners)

Dimension 16-21 3D head pose in camera coordinate system (rotation and translation, as explained in [44]).

Dimension 22-24 face center in the camera coordinate system

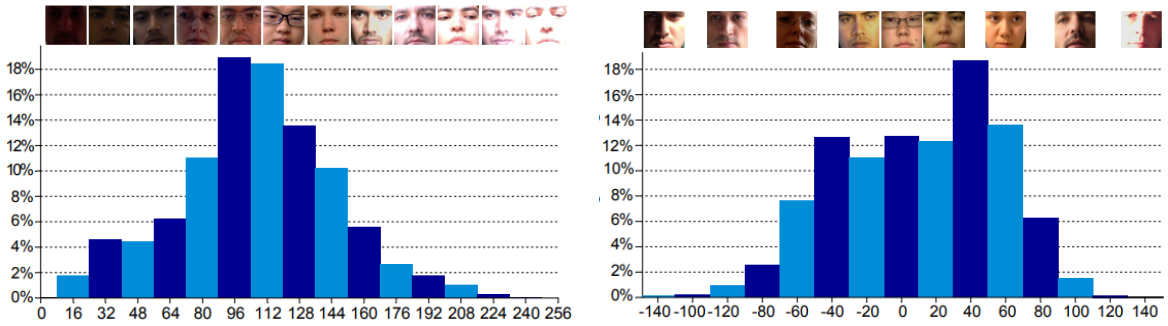
Dimension 25-27 3D gaze target location in the camera coordinate system

Dimension 28 which eye is used for the evaluation subset in [53]

My target is to predict 3D gaze target location (dimension 25-27).

Data distributions

The MPIIFaceGaze dataset has a wide variety of illumination conditions. This fact can be seen from figure 5.6a, where the greyscale intensity follows the normal distribution. On top of this, the position of the source of light also varies a lot on the horizontal axis. There is about a uniform distribution between left versus right side lightning conditions 5.6b.

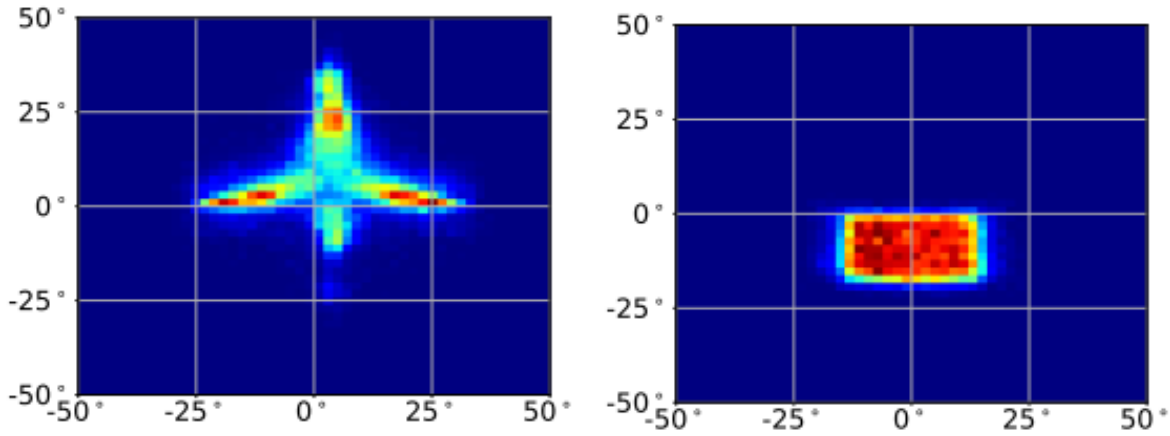


(a) Mean greyscale intensity

(b) Horizontal difference in mean greyscale intensity

Figure 5.6: Lightning conditions distributions (in percentages) of the MPIIGaze-Dataset [55]

When looking at the head angle distributions in figure 5.7a I see that it follows a specific pattern. People tend to move their head when they need to focus the bottom-left and bottom-right corners, as well as looking straight up. Nevertheless, looking over gaze angle distribution in figure 5.7b, the yaw takes uniform values between $-18°$ and $18°$ and the same can be stated for pitch angles between $-1.5°$ and $-20°$.



(a) Head angle distribution

(b) Gaze angle distribution

Figure 5.7: Distribution of head angle and gaze angle in degrees (yaw-pitch representation) [55]

Data pre-processing

Following the work of Ahmed A. Abdelrahman et al. [4] I want to predict the 3D gaze direction in spherical coordinates (yaw and pitch). In order to obtain these coordinates I have used the pre-processing scripts proposed by Yihua Cheng et al. [1]. Their approach is described in Figure 3.3. In short, they are rotating the virtual camera such that the z-axis is pointing towards the subject, then it rotates it such that the person's head is straight and finally the image is scaled such that the same distance between the subject and the virtual camera is ensured.

After using the rectification method mentioned before, the 3D vector I want to predict is defined as having one of its axis perpendicular to the screen plane. Predicting a value for this axis is not valuable to us, because it is often the same with a constant value (given by the normalized distance between the subject and the camera). That is why I chosed to only predict the other 2 axis by transforming the 3D vectors in pitch/yaw rotation movements. As a result, I get normalized face RGB images and normalized labels for 2D Gaze angles (yaw and pitch). On top of that, Ahmed A. Abdelrahman et al. [4] also resize images to 448x448 such that they have the same size as those in the Gaze360 dataset and I follow the same procedure such that I can replicate their results.

Evaluation strategies

First of all, I implemented 3 different evaluation strategies based on different train/test splitting options for the MPIIFaceGaze dataset (15 subjects):

1. **Single model leave-one-subject-out evaluation** (train one model on 14 subjects and test on the remaining subject)

2. **Multiple models leave-one-subject-out evaluation** (train 15 models each on 14 different subjects and test on the remaining subject)

3. **Single model leave-three-subjects-out evaluation** (train one model on 12 subjects and test on the remaining 3 subjects)

The first strategy is used when I need to reduce training time, the second one provides a better evaluation as I train one model for each leave-one-subject-out combination, but it runs really slow. The third strategy is better at providing an unbiased evaluation of the model, while still being as fast as the first strategy.

When deciding on the train/test split strategy the next idea is very important: one should never randomly mix images from different subjects and then split them between train and test set. This is a well-known test data leakage, because the model

already saw images of all the subjects, and I can not evaluate how the model performs on unseen subjects.

5.3 Baseline: L2CS model reproduction

In this section I denote my baseline. I have reproduced the model introduced by Ahmed A. Abdelrahman et al. [4]. It is based on the ResNet convolutional neural network architecture which I described in Section 4.2. The authors feed the input feature map into the backbone (ResNet50) which slightly reduce the image size after each layer. Then an average is taken across the width and height dimension to remain with only the batch and channel dimensions using an adaptive 2d average pooling layer. At this moment, the feature map is liniarized and fed into two fully connected layers one for yaw binned prediction and one for pitch binned prediction. After each convolution in the original ResNet architecture a batch normalization layer and a relu activation function are used. Architecture is shown in Figure 5.8.

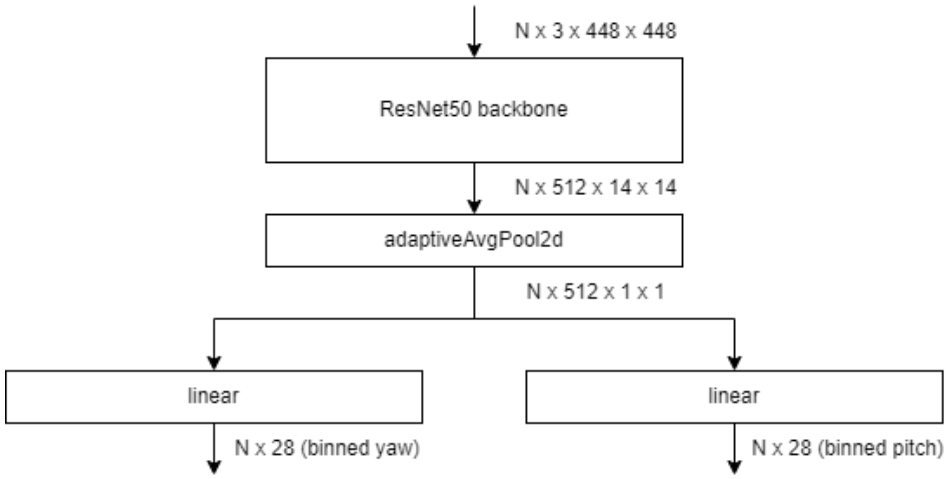


Figure 5.8: L2CS model architecture

The output of the model represents 28 gaze angles bins (for each yaw and pitch) between -42° and 42° (e.g. first value represents value -42°, second values represents -39°, and the last one represents 42°. These 28 values are fed through a softmax layer in order to transform unconstrained values into probabilities. The intuition behind this is, each value represents the probability that the gaze angles coresponds to that specific bin. These probabilities are used for computing the Cross-Entropy Loss. Based on the probability of each bin, a mean angle value is computed which stands as the final gaze angle prediction. This value is then used in the computation of the Mean-Squared Error loss.

Loss function

The overall loss function is defined in Equation 5.1 where y is the groundtruth and the $\hat{y}$ is the model prediction. The loss function has two components: the Cross-Entropy Loss (CEL) and the Mean Squared Error (MSE) defined in Equation 5.2 and 5.3 respectively. The Cross-Entropy Loss returns an error based on the discrepancy between the predicted bin and the true bin class. The Mean Squared Error on the other hand, is based on the Euclidean distance between the predicted gaze vector and the groundtruth.

For instance, lets consider a groundtruth of -42° and a prediction of 0.9 for the correct bin (bin 0) and 0.1 for bin 1. The CEL value is $-1 * log(0.9) = 0.045$. The final gaze angle prediction is $(-42) * 0.9 + (-39) * 0.1 = -41.7$. The MSE value is $0.3^2 = 0.09$. And finally the total loss is $0.045 + 0.09 = 0.135$. On the other hand, if I change the prediction to 0.5 for the correct bin and 0.5 for bin 1. The CEL value is $-1 * log(0.5) = 0.3$. The final gaze angle prediction is $(-42) * 0.5 + (-39) * 0.5 = -40.5$. The MSE value is $1.5^2 = 2.25$. And finally the total loss is $0.3 + 2.25 = 2.55$.

$$L(y, \hat{y}) = \text{CEL}(y, \hat{y}) + \text{MSE}(y, \hat{y}) \tag{5.1}$$

$$\text{CEL}(y, \hat{y}) = -\sum_{i} y_i \log \hat{y}_i \tag{5.2}$$

$$\text{MSE}(y, \hat{y}) = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 \tag{5.3}$$

Training setup

I trained the model exactly as explained in the work of Ahmed A. Abdelrahman et al. [4]. The model has a pretrained ResNet-50 backbone with the initial layers frozen. I trained it on the MPIIFaceGaze dataset. I used the first folding strategy explained in Section 5.2 testing on subject 2 (fold 2). The training set contains 42000 samples (3000 samples per subject, for 14 subjects) while the test set contains 3000 samples (1 subject). It was trained with batch size 16, learning rate 0.00001, using the Adam optimizer for 50 epochs. The experiment ran remotely on a NVIDIA GeForce RTX 3090 graphics card with 24 GB video memory. For further reference I will consider this model my baseline.

Training results and evaluation

The experiment ran for 10 hours and 10 minutes. I have visualized the loss curves in Figure 5.9. The loss is decreasing fast until it reaches a plateau. I can see that the

model performs a bit worse on the test set, which is normal since it is predicting based on unseen data.

I used the angular error defined in Section 3.3 as my evaluation metric. This metric is computed on the test set after each training epoch and can be observed in Figure 5.10. The metric as well as the test loss decreased until epoch 20 where the learning starts to slow down and no further progress can be seen on the angular error metric. I also added 12 random test samples with predictions and groundtruths at different timestamps in the training proccess (before training, after 5/20/50 epochs of training) for visual evaluation. See Figure 5.11. Once again, I can identify great improvements from epoch 1 to 20 but not as much from 20 to 50 epochs.

Results comparison

Comparing my results after 50 epochs (4.5° angular error) to the L2CS-Net reported error of 3.8° on subject 2 I find that my solution has an 18% decrease in performance. However, I consider the experiment a success, because during training I also reached 4.0° angular error (on epoch 46) and I assume that with a better seed or by selecting the best performing model during training I would have reached a similar performance.

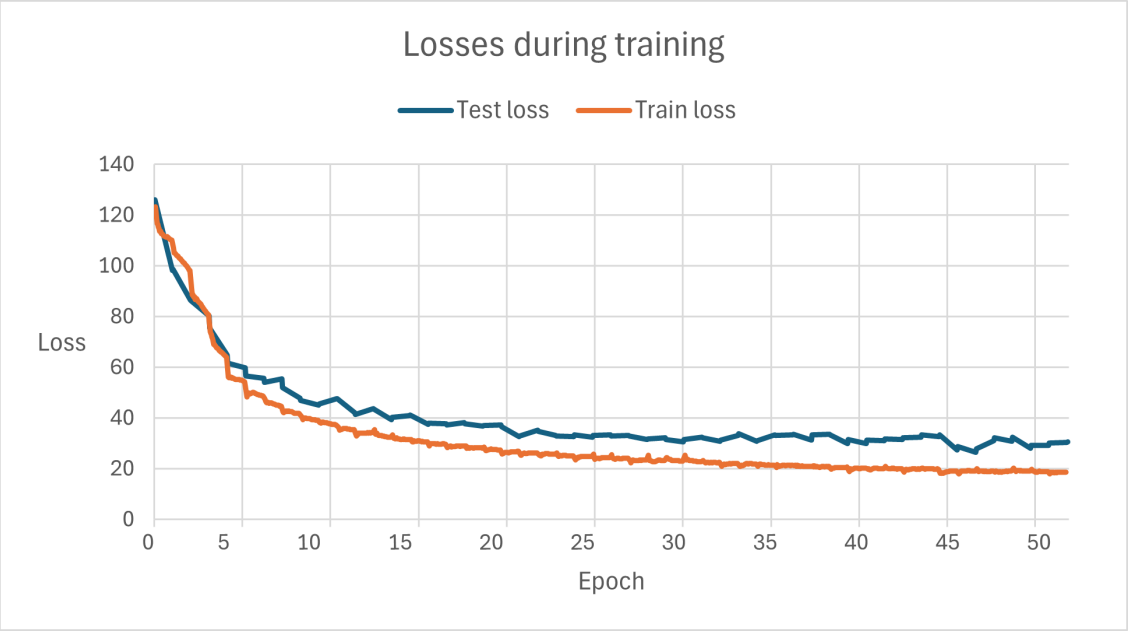


Figure 5.9: Losses during the training process of the reproduced L2CS-Net model (**Baseline**). ResNet-50 backbone. Trained on the MPIIFaceGaze dataset with 42000 training samples and 3000 testing samples. Batch size 16. Learning rate 0.00001. 50 epochs. Adam optimizer. Trained on an NVIDIA RTX 3090. See Section 5.3 for more information on the configuration.

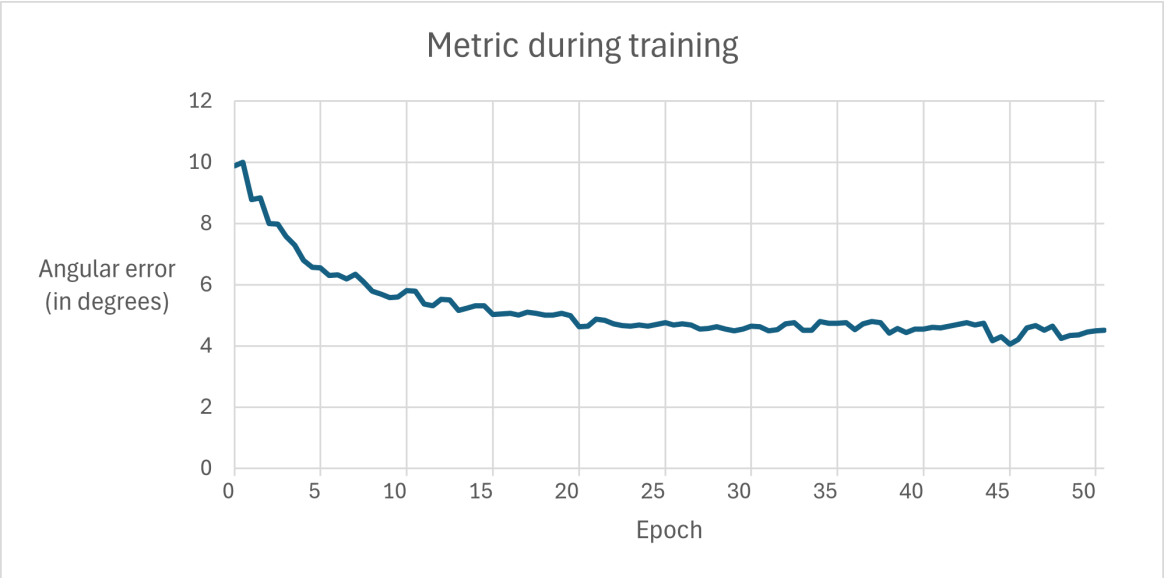


Figure 5.10: Angular error on test set during the training process of the reproduced L2CS-Net model (**Baseline**). ResNet-50 backbone. Trained on the MPIIFaceGaze dataset with 42000 training samples and 3000 testing samples. Evaluation on subject 2. Batch size 16. Learning rate 0.00001. 50 epochs. Adam optimizer. Trained on an NVIDIA RTX 3090. See Section 5.3 for more information on the configuration.

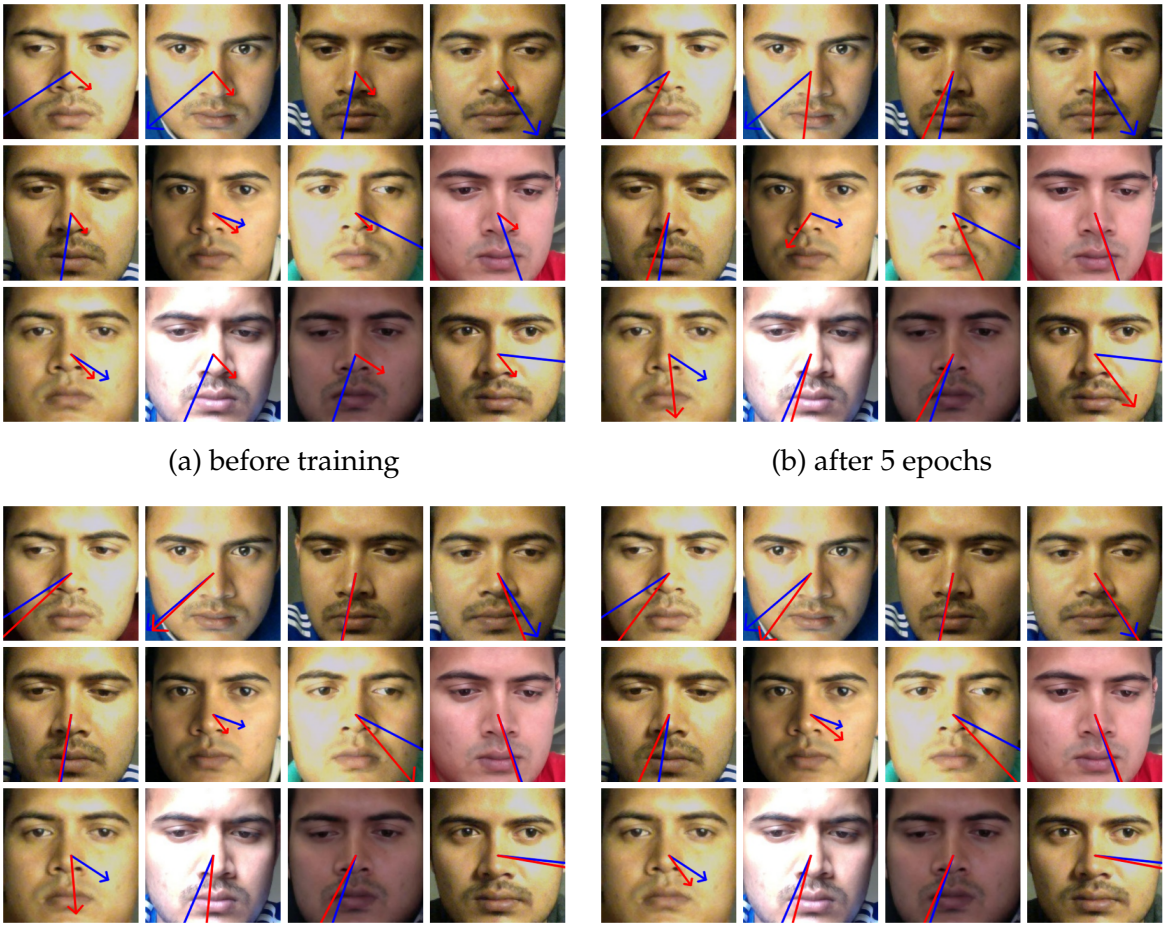
Tuning overview

In this section, I want to experiment using different hyper-parameters for the Baseline model described in Section 5.3. I am going to explore the effects of changing the learning rate, the batch size, the number of epochs, the backbone's number of layers, and the seed used to make the experiment random and reproduceable.

At the beginning I needed to reduce the computational cost of training and I only trained for 5 epochs. This low number of epochs was not enough to demonstrate model performance, and so I decided to train on another computer with more computing power. On this machine I experimented with 10-50 epochs on each run. The number of epochs I trained for can also be seen in Table 5.1.

I am going to perform a manual tuning of these hyperparameters. After experimenting with one hyperparameter I will first filter out models that showed signs of overfitting or underfitting and then select the best option based on the angular error metric.

The results of these experiments can be observed in Table 5.1. The first row of the table is defined by the initial run of the Baseline (the reproduced L2CS-Net) described in Section 5.3 along with its hyperparameters. For the next experiments I will label new versions of the model such that I can reference them later. The columns represent the epochs, mean angular error, train loss and test loss for each run.



(a) before training

(b) after 5 epochs

(c) after 20 epochs

(d) after 50 epochs

Figure 5.11: Labeled random samples from the test set. Blue arrow is the groudtruth, red arrow is the prediction. Visual evaluation on the test set during the training process of the reproduced L2CS-Net model (**Baseline**). ResNet-50 backbone. Trained on the MPIIFaceGaze dataset with 42000 training samples and 3000 testing samples. Evaluation on subject 2. Batch size 16. Learning rate 0.00001. 50 epochs. Adam optimizer. Trained on an NVIDIA RTX 3090. See Section 5.3 for more information on the configuration.

Rationale for model tuning

I first tried 3 types of backbones ResNet-50, ResNet-34, and ResNet-18. I selected the model with the ResNet-18 backbone (model M2) since there was not a significant loss in performance, but the training time halved. Next I explored different learning rates. I first decided to use a higher learning rate of 0.0001 (model M3) since it showed better perfomance. Next I explored with different batch sizes. I selected batch size 64 (model M4), since I thought it might reduce overfitting, However when I tried to train for different numbers of epochs and I actually overfitted starting as low as 10 epochs.

At this point I concluded that using a higher learning rate was a mistake and I continued using the initial learning rate of 0.00001 (model M2). I also discovered that using only 5 epochs of training is not enough to identify overfitting/underfitting so I decided to use 10 epochs from now on. Once again I compared different batch sizes, and selected batch size of 64 (model M6) based on the idea that this model will perform best in a longer training session. Then I tried different random seeds, which showed that there is quite a big influence on the model performance (of almost 1° on the mean angular error). I selected the model that performed best (model M7). Finally, I compared this model's performance when trained for different number of epochs. As one can see in Table 5.1 model M8 did not overfit as did the M5 model.

Tuning conclusion

I have shown that one can achieve similar performance as the baseline model with ResNet-50 backbone that was trained 50 epochs with a simpler architecture such as a ResNet-18 with as low as 10 epochs of training. One interesting fact is that the M3 model which has a high learning rate and was only trained on 5 epochs has managed to show the best performance of all the experiments, however I consider this model to be unstable, because of its extreme hyperparameters. I argue that the best performing stable model is M8 with a small decrease in performance (0.3° angular error) compared to the initial baseline model. The tuned M8 model has the advantage that it can be trained 60% faster over the baseline. I will call the tuned M8 model my optimized L2CS-Net ResNet-18.

5.4 New approach: FastSightNet

In this section I present my new model called FastSightNet. The motivation behind it is that I need a lightweight model, with fast inference time such that it can work on any low resources hardware. Another reason was that training the L2CS

ResNet model is quite computational intensive and I believe that other more efficient models will work in this scenario at least as well as L2CS-Net. My model is based on the MobileNet architecture described in Section 4.2. The FastSightNet architecture can be observed in Figure 5.12. The input of size 3x224x224 is fed through the MobileNetV2 backbone. The output is fed to an adaptive average pooling layer to reduce the feature-map size to Nx1280x1x1 similar to the L2CS-Net model defined by Ahmed A. Abdelrahman et al. [4]. Finally, the feature-map is liniarized and then goes through a fully connected layer and then outputs 2 values (pitch and yaw angles) for each image in the batch.

Training setup

I chose to freeze the first 9 layers of the backbone, since it was trained on ImageNet dataset and it already learned most of the small patterns, however I train the rest of the backbone and the additional layers on the MPIIFaceGaze dataset. I used the first folding strategy explained in Section 5.2 testing on subject 10 (fold 10). The training set contains 42000 samples (3000 samples per subject, for 14 subjects) while the test set contains 3000 samples (1 subject). It was trained with batch size 32, learning rate 0.0001, using the Adam optimizer for 5 epochs. The experiment ran remotely on an NVIDIA GeForce GTX 1060. The loss I used is L2 loss for both yaw and gaze.

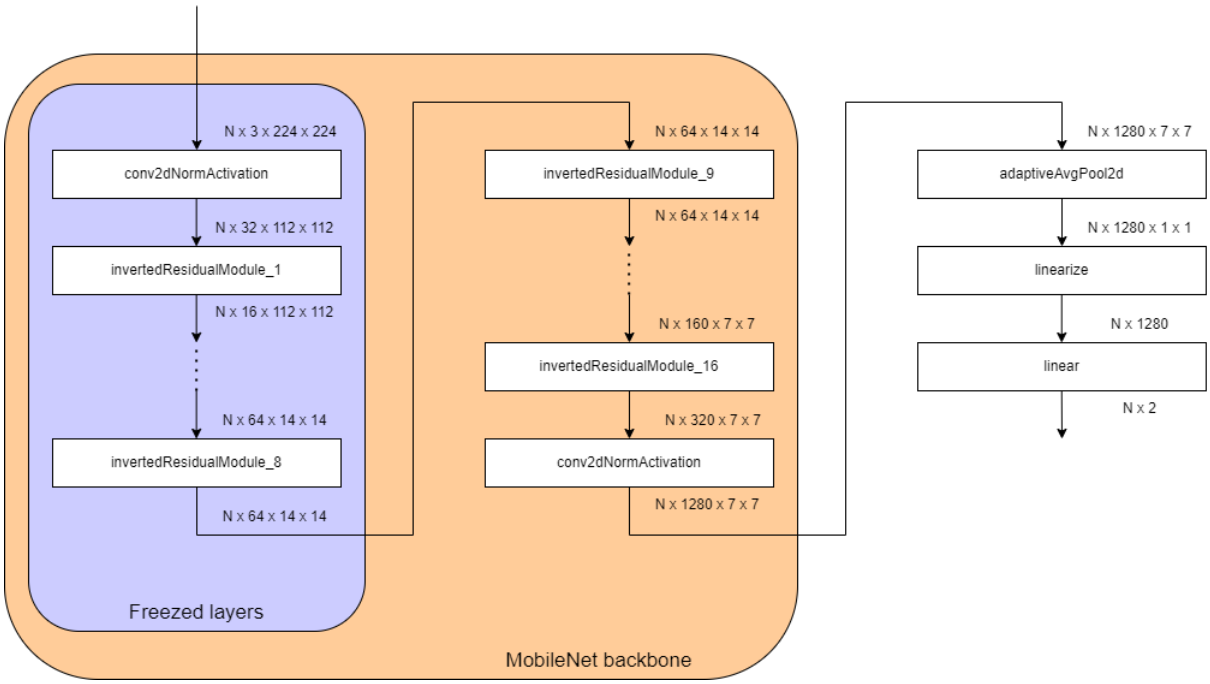
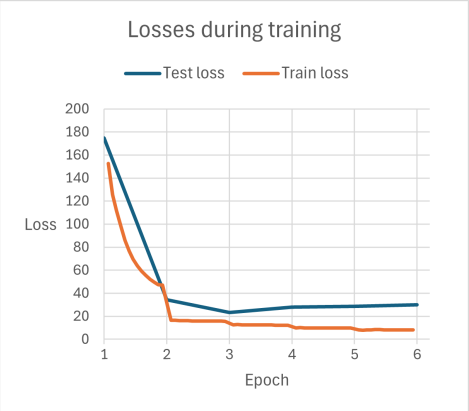


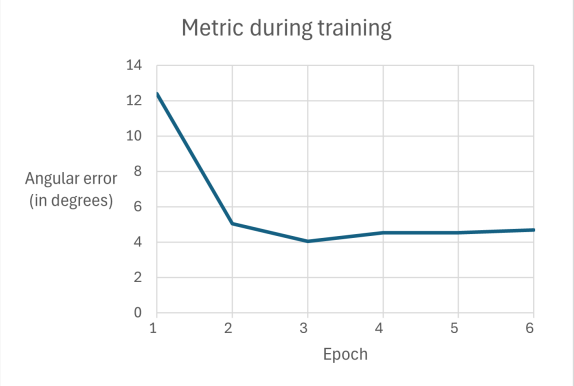
Figure 5.12: FastSightNet model architecture

Training results

The experiment took 19 minutes to run. Losses are decreasing fast until convergence, see Figure 6.8a. Last angular error evaluation of the model shown that it reached 4.6° angular error on the test set as one can see in Figure 6.8b.



(a) Train and test losses.



(b) Angular error on test set.

Figure 5.13: Training results of the FastSightNet model (**Initial model**). MobileNetV2 backbone. Trained on the MPIIFaceGaze dataset with 42000 training samples and 3000 testing samples. Evaluation on subject 10. Batch size 32. Learning rate 0.0001. 5 epochs. Adam optimizer. Trained on an NVIDIA GeForce GTX 1060. See Section 5.4 for more information on the configuration.

Tuning overview

Tuning

In this section I will experiment with different hyperparameters for my FastSightNet model. I are going to explore changing the batch size, the learning rate, the seed, the number of epochs, and the test subject. Initially I kept the number of epochs low (5 epochs), to be able to perform many experiments in a shorter amount of time, but after selecting the final model I continued to evaluate it on 10 epochs of training. The results of these experiments can be observed in Table 5.2.

Rationale for model tuning

The initial run denotes the run of the FastSightNet explained in Section 5.4. Firstly, varying the learning rate I observed that a larger learning rate (0.001 and 0.0005) leads to overfitting since the training loss is very low compared to the test loss, that's why I decided to continue with the smallest learning rate (0.00001) I have tried (model F2).

While experimenting with different batch sizes I observed that a large batch size means the algorithm will take small steps such that it optimizes all elements in the batch, which means it will learn slowly. This effect can be seen from the high losses for both training and testing. On the opposite low batch size means there are going to be more weight updates which leads to a faster convergence in the same amount of epochs. For instance, batch size 16 leads to 33.428 test loss while a batch size of 128 leads to 63.184 test loss. I decided to move forward with the initial batch size of 32. Even if this did not provide the best perfomance over all, it shows signs of better convergence which is beneficial for the moment I will train for more epochs.

Next I experimented with 10 epochs of training which showed better results, so I selected model F3. Influence of randomness seed shown a possible difference of 0.5° in mean angular error however model F3 performed best. Finally, changing the test fold is not intended to be used in a comparison since test data is completely different, however one can see that the model performs similarly well on other subjects too.

Tuning conclusion

Now that I have compared different runs, I can say that the initial model was overfitting, since the train loss is really low compared to the test loss. An interesting model was F4 which shown the lowest angular error of 4.1° and also a bit of an overfitting, however it found a way to minimize the loss better than any other model reaching 23.768 loss. These models are not stable enough to be considered in a reliable setup. My final F3 model, shown similar performance with the initial model with a mean angular error of 4.6°. The advantage of the tuned F3 model is that it is more stable, since it did not overfit to the training data and I expect it to perform better on unseen data. I will call the tuned F3 model FastSightNet from now on.

5.5 Model comparison: L2CS-Net & FastSightNet

Now that I have tuned the model I want to see how well it performs on leave-one-subject-out cross-validation. What it means is that I train one model for each combination of 14 out of 15 subjects and test it on the remaining subject. Results can be seen in Figure 5.14. It can be observed that FastSightNet achieved the best performance on subject 0 with 3.14° angular error. The performance trend is comparable with the L2CS-Net model. One can also identify that my model performed better than L2CS-Net on subject 11. Overall, I achieved a mean angular error of 5.1° compared to L2CS-Net error of 3.9°, resulting in a decrease in performance of 1.2°.

The advantages of the FastSightNet model are the inference time, training time, and model size. See Table 5.3. The FastSightNet shows a 70% decrease in inference

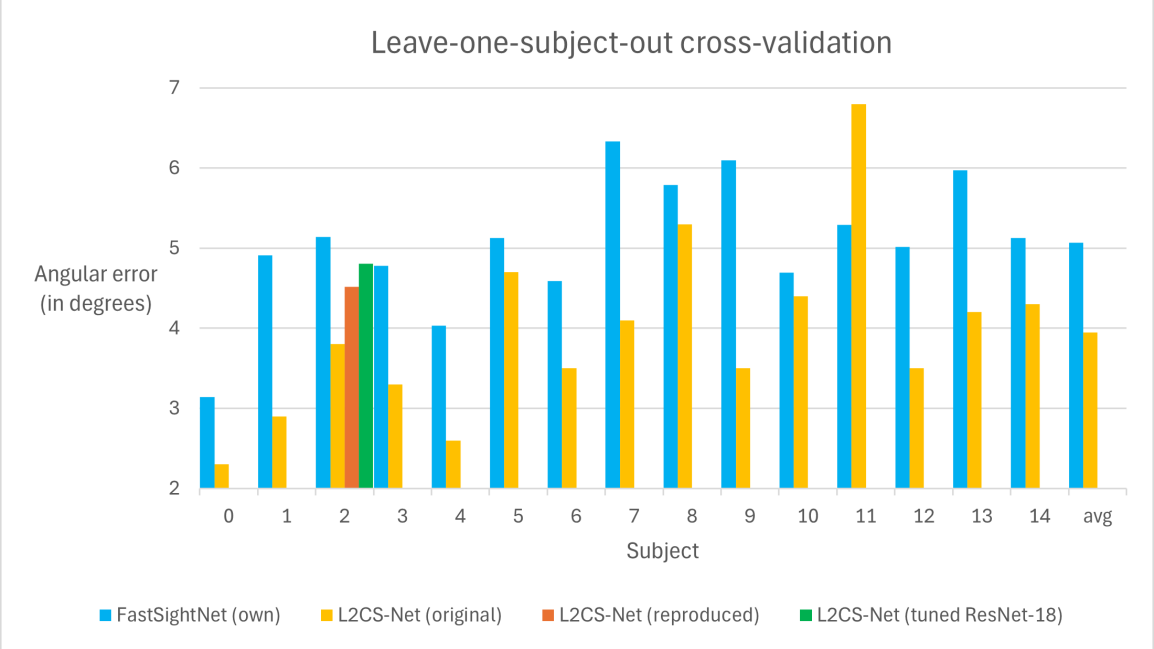
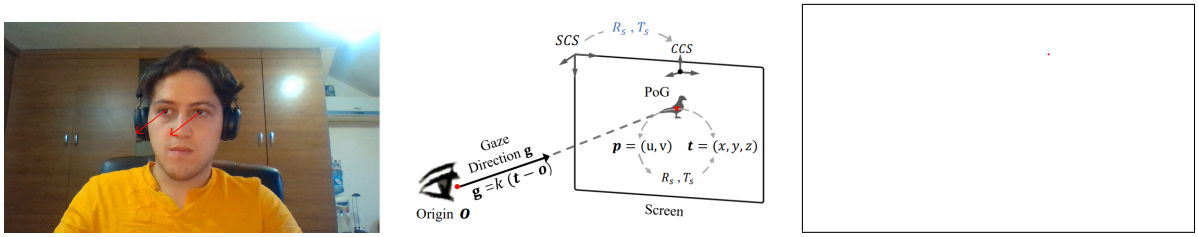


Figure 5.14: Cross-validation comparison between FastSightNet and L2CS-Net models. FastSightNet is the tuned version of my model based on MobileNetV2 backbone. It was trained for 10 epochs, batch size 32, learning rate of 0.00001. The original L2CS-Net proposed by Ahmed A. Abdelrahman et al. [4] as well as the version reproduced by me are based on a ResNet-50 backbone. They were trained for 50 epochs, batch size 16, and a learning rate of 0.00001. My optimized version of L2CS-Net is based on a ResNet-18 backbone and it was trained for 50 epochs, with a batch size of 64, and a learning rate of 0.0001.

time, 95% decrease in training time, as well as a 90% decrease in the number of parameters compared to the reproduced L2CS-Net model. FastSightNet exceeds my goal of 60 frames per second which makes it usable in a day-to-day work flow from the inference time point of view. On the other hand, I consider the reproduced L2CS-Net to be not ideal for a real-time usecase since it only runs in 24 frames per second. As one can see there is a trade-off between the accuracy of the model and the inference time. For instance, FastSightNet shown a decrease in performance of 30% but on the other hand the inference time improved by over 70%.

5.6 Data post-processing methods

The main use-case of the gaze estimation model is to use it as an human-computer interface. For instance, one can aspire to move the cursor on the screen as indicated by the gaze estimation model. In order to perform this action, one must translate from 3D output vectors cu 2D points of gaze on the screen. Inspired by the work of Yihua Cheng et al. [10] I reproduce their methods in my case. Firstly, the gaze vectors are intersected with the screen plane. I assume that the person's face is placed centered on the camera at a distance of 50 cm, and also perpendicular to the screen. After performing the intersection I get a point estimation in the camera coordinates system. Now I would like to the gaze point relative to the top-left corner of the screen. I used the hard-coded position of camera relative to the top-left corner of the screen and performed a rotation such that I can express the gaze point in the screen coordinates system. I used millimeters as units, however the conversion into pixels was not a difficult problem. A diagram of this method can be observed in Figure 5.15b.



(a) 3D gaze vector representation

(b) 3D-to-2D conversion diagram [10].

(c) 2D screen coordinates representation

Figure 5.15: Conversion of 3D gaze estimation output to 2D points in the screen coordinates system.

Model description	Epochs	Mae	Train loss	Test loss
Baseline 5.3 (M1)	50	**4.515**	18.594	30.656
M1 backbone=ResNet-50	10	5.603	38.205	45.526
M1 + backbone=ResNet-34	10	6.112	37.834	52.074
M1 + backbone=ResNet-18 (M2)	10	**5.376**	38.041	42.576
M2 + lr=0.000001	5	8.583	109.259	94.525
M2 lr=0.00001	5	6.415	55.616	59.956
M2 + lr=0.0001 (M3)	5	**4.475**	24.718	31.715
M2 + lr=0.001	5	4.754	22.190	36.277
M3 + batch=4	5	5.559	35.051	43.508
M3 + batch=8	5	4.753	27.290	35.087
M3 batch=16	5	**4.475**	24.718	31.715
M3 + batch=24	5	4.710	23.769	35.037
M3 + batch=32	5	5.894	23.352	49.059
M3 + batch=64 (M4)	5	5.083	24.696	38.725
M4 epochs=5	5	5.083	24.696	38.725
M4 + epochs=10	10	5.127	18.742	38.286
M4 + epochs=25	25	**4.992**	13.027	36.595
M4 + epochs=50 (M5)	50	5.542	10.202	43.478
M2 batch=16	10	**5.376**	38.041	42.576
M2 + batch=32	10	5.459	40.121	46.277
M2 + batch=64 (M6)	10	6.244	48.936	57.164
M6 seed=1	10	6.244	48.936	57.164
M6 + seed=2	10	6.999	57.923	71.022
M6 + seed=3 (M7)	10	**5.946**	50.841	53.086
M7 + epochs=5	5	8.368	83.962	92.685
M7 epochs=10	10	5.946	50.841	53.086
M7 + epochs=25	25	5.144	30.049	39.698
M7 + epochs=50 (M8)	50	**4.803**	22.061	35.029

Table 5.1: Tuning experiments of the reproduced L2CS-Net model (**Baseline**). ResNet-50 backbone. Trained on the MPIIFaceGaze dataset with 42000 training samples and 3000 testing samples. Evaluation on subject 2. Batch size 16. Learning rate 0.00001. 50 epochs. Adam optimizer. Trained on an NVIDIA RTX 3090. See Section 5.3 for more information on the configuration. Each row shows the change that was made to obtain the new version of the model.

Model description	Epochs	Mae	Train loss	Test loss
Initial model 5.4 (F1)	5	**4.675**	8.282	30.041
F1 + lr=0.00001 (F2)	5	5.163	25.758	36.414
F1 + lr=0.00005	5	4.398	11.343	27.234
F1 lr=0.0001	5	4.675	8.282	30.041
F1 + lr=0.0005 (F4)	5	**4.120**	6.625	23.768
F1 + lr=0.001	5	5.107	7.508	34.808
F2 + batch=16	5	**4.918**	24.616	33.428
F2 batch=32	5	5.163	25.758	36.414
F2 + batch=64	5	5.580	32.351	24.149
F2 + batch=128	5	7.107	67.436	63.184
F2 epochs=5	5	5.163	25.758	36.414
F2 + epochs=10 (F3)	10	**4.696**	16.376	30.741
F3 seed=1	10	**4.696**	16.376	30.741
F3 + seed=2	10	4.831	16.173	32.065
F3 + seed=3	10	5.191	16.169	36.473
F3 + fold=2	10	5.141	16.261	33.165
F3 + fold=5	10	5.127	16.201	33.771
F3 fold=10	10	**4.696**	16.376	30.741
F3 + fold=14	10	5.128	16.879	31.789

Table 5.2: Tuning experiments of the FastSightNet model (**Initial model**). MobileNetV2 backbone. Trained on the MPIIFaceGaze dataset with 42000 training samples and 3000 testing samples. Evaluation on subject 10. Batch size 32. Learning rate 0.0001. 5 epochs. Adam optimizer. Trained on an NVIDIA GeForce GTX 1060. See Section 5.4 for more information on the configuration. Each row shows the change that was made to obtain the new version of the model.

Model	Inference time	Training time	Parameters
L2CS-Net (reproduced)	0.041s (24 fps)	10h10min	23,628,932
L2CS-Net (tuned ResNet-18)	0.017s (59 fps)	3h45min	11,206,788
FastSightNet (own)	**0.013s (77 fps)**	**30 min**	**2,226,434**

Table 5.3: Model metrics comparison between FastSightNet and L2CS-Net models. FastSightNet is the tuned version of my model based on MobileNetV2 backbone. It was trained for 10 epochs, batch size 32, learning rate of 0.00001. The original L2CS-Net proposed by Ahmed A. Abdelrahman et al. [4] as well as the version reproduced by me are based on a ResNet-50 backbone. They were trained for 50 epochs, batch size 16, and a learning rate of 0.00001. My optimized version of L2CS-Net is based on a ResNet-18 backbone and it was trained for 50 epochs, with a batch size of 64, and a learning rate of 0.0001.

Chapter 6

GazeTrack: System development

I am going to devise a system that can be used to showcase the performance of the gaze estimation model I trained myself. I will follow the normal flow of software development. I will start by analyzing functional and non-functional requirements, and then continue with the application design. In the next section, I will talk about implementation details, and how I design tests for my system.

6.1 Requirements

Simulation: Collecting requirements from client

Application main features are showcasing the model performance in real life scenarios. The app should have 2 main features: show the camera feed enhanced with gaze vectors, and show a white canvas with random blue points appearing at regular intervals and red dots representing the predicted gaze points. The application should run in real-time and not stutter because of the model's inference time. Another required feature is to show inference time in fps on the screen. The app should only use the camera when the the camera feed is actually needed (e.g. not in the menu screen).

Functional and non-functional requirements

I can split these requirements in two groups: functional requirements, which denote actual functionalities required for the the application, and non-functional requirements which refer to constraints and performance characteristics the app should adhere to.

Functional requirements:

F1 Show the camera real-time feed enhanced with gaze vectors.

F2 Show the white canvas with targets and predicted gaze points.

Non-functional requirements:

NF1 Application should not stutter.

NF2 Application should only use camera feed when it is actually needed.

Requirements phase defect

I identified some errors in specifying the application requirements:

1. How should the camera feed/canvas be shown? Full-screen or a pop-up window? (requirements are missing) Solution: camera feed and canvas should be shown in full-screen.

2. How often should the random blue points be generated. (requirements are incomplete) Solution: they should be regenerated every 5 seconds.

3. In what state should the program be at beginning? (initial system state has not been considered) Solution: the menu window should be opened at the beginning.

6.2 Application design

Usecases

The design of my application is driven mainly by the two usecases. I define one of the use-cases using its description schema in Table 6.1, as well as its UML Sequence Diagram shown in Figure 6.1. Finally I split the whole application into subsystems such that I increase cohesion and decrease coupling between classes. The subsystem diagram can be observed in Figure 6.2

6.3 Implementation

The application I want to develop should allow a fast integration of the machine learning gaze estimation models trained in the previous chapter, so I used Python to also build the application. The app is going to be built with Model-View-Controller (MVC) architecture in mind. The model is represented by camera frames and gaze vector predictions, the views are represented by the GUI windows, while the Gaze Predictor and Camera classes act as controllers that redirect data to the views.

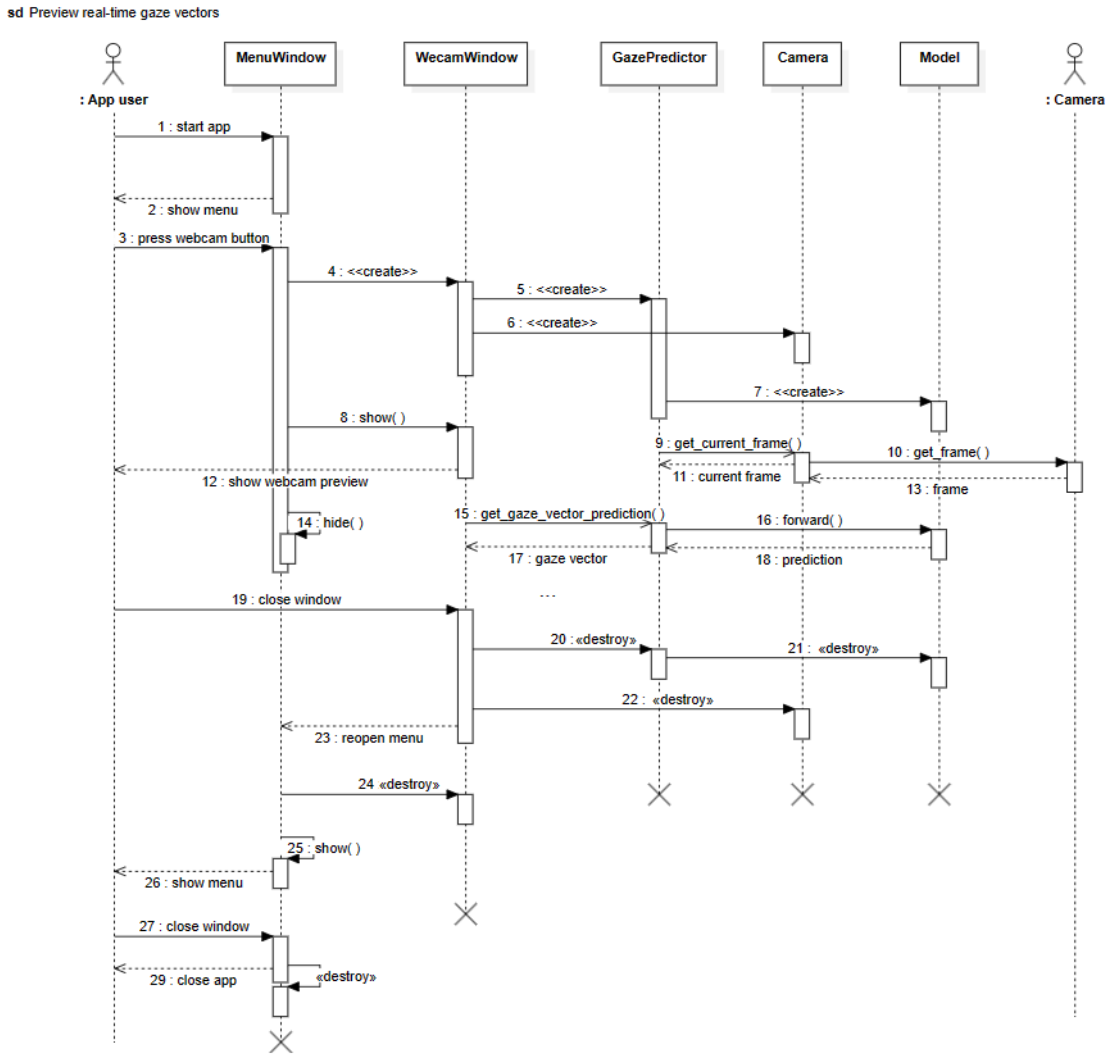


Figure 6.1: UML sequence diagram for usecase I

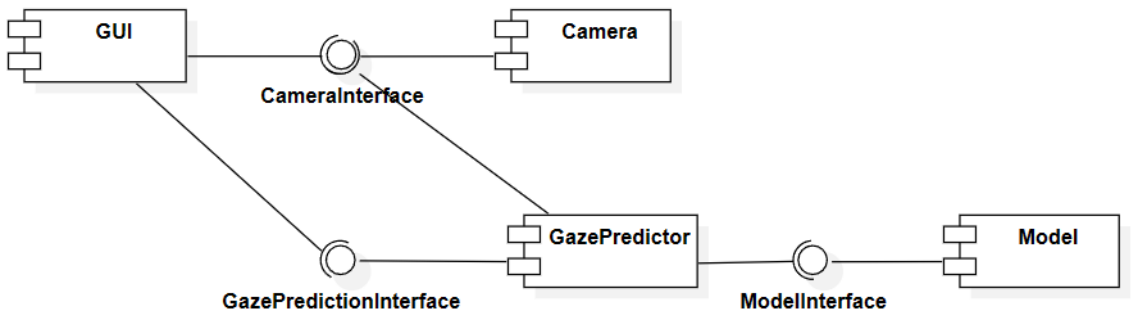


Figure 6.2: UML subsystem diagram

Usecase name	Preview real-time gaze vectors.
Actor	Application user.
Event flow	1. User presses the 'webcam' button. 2. The systems opens the camera device, shows the preview of the camera, and shows gaze vector prediction of the gaze estimation model and the fps counter of the inference in top left corner. 3. The user changes his point of gaze. 4. The system shows new gaze vector prediction. 5. User pressed the exit button. 6. The system closes the camera and the preview window and shows the menu again.
Input conditions	The user opened the app and sees the menu window.
Exit conditions	The system's camera is closed and the user is back in the menu window.
Quality requirement	The app should not be blocked by the time-consuming action of the model's inference.

Table 6.1: Description schema for usecase I

Libraries

The system does not require a complex UI design so I opted to use PyQt5 framework [31]. Qt is a cross-platform framework used for developing graphical user interfaces. Qt offers access to the entire Qt library with native look and feel across multiple platforms. Qt has an extensive library of widgets and it's own event handling mechanism featuring Signals and Slots. I have used signals and slots to communicate between GUI windows as well as producer workers. For camera integration I used OpenCV [47], for constructing my model I used the PyTorch framework [11], and for concurrent execution I used the threading library.

Data flow

PyQt as the majority of GUI frameworks has a drawing thread (main thread). As I know model inference is a time-consuming task, so I am not running it on the main thread. When constructing a Camera or a Gaze Predictor class, two additional threads are created, one that captures new camera frames and saves in a local variable, and one that takes the camera frame available in the camera class and runs the model inference on it, saving it as the most recent prediction. Finally, two producer workers are set to recurrently take image frames from the camera as well as gaze predictions from the gaze predictor and show them on the screen.

Concurrency problems

As I have presented the systems earlier, two thread may access the camera's locally saved frame simultaneously and race conditions may occur. In order to solve this problem I am using locks (mutexes) in order to access the data in a safe way. Although I am using multiple threads in my system that run concurrently, Python limits the parallel execution of these threads. In fact at most one thread is running at any given moment, because of the Global Interpreter Lock (GIL). This interpreter exists within CPython interpreter and its role is to ensure consistent multi-threaded memory management.

Design patterns

One problem I encountered was that both a camera producer worker and the gaze predictor instantiated a camera entity, which is not allowed. To ensure this won't happen I decided to use the Singleton design pattern. It allows only one entity of an object to be created. However, using this pattern for the camera class, it would not shut down as requested in NF2 requirement when not in used. To also solve this issue, I implemented a reference counter for the Singleton object which destroys it if no other object references it.

6.4 Testing

I have tested the application at two levels: unit testing and integration testing. For instance, I created unit tests for the Gaze Predictor class replacing the camera and the model object it uses with mock objects. I have tested that the instantiation of the class happens only once (respecting the Singleton design pattern). I also tested that the respective methods of both Camera and Model are called when the main functionalities of the class are used. I replaced the locking mechanism used for multithreading safety measures with a mock and checked that the acquire and release functions are called when any of the class methods are used.

Regarding integration tests, I tested the integration of the machine learning model class within the Gaze Predictor class. I wrote tests that cover the cases when the Model class receives invalid arguments and throws FileNotFoundError, when it receives a fully dark image and no face can be detected and it should return None, and finally when it receivs a face image and it returns the position of the eyes and the gaze vector.

6.5 User guide

When starting the app a menu will appear (See Figure 6.3). One can select the gaze estimation model to be used and the type of demo. After selecting the webcam demo, a preview of the real-time webcam feed will appear, enhanced with 3D gaze vectors (See Figure 6.4). In the top left corner the inference rate is shown. After selecting the tracing demo, a white canvas and a target (blue dot) will appear on the screen (See Figure 6.5). 3D gaze prediction is converted to 2D and the gaze point (red dot) is shown on the screen.

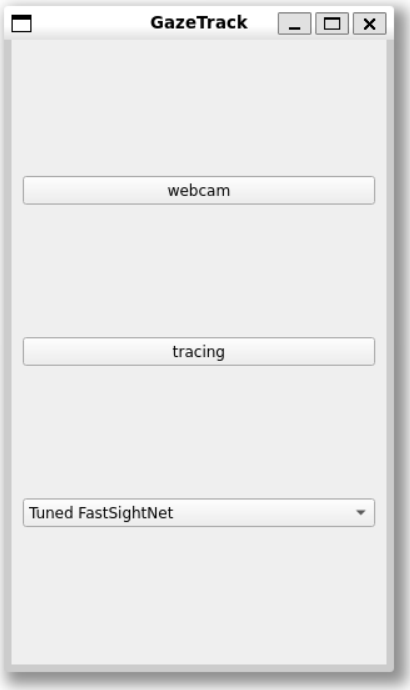


Figure 6.3: The menu of the GazeTrack system, including 2 buttons that open up the demos and a dropdown menu from where one can select the gaze estimation model.

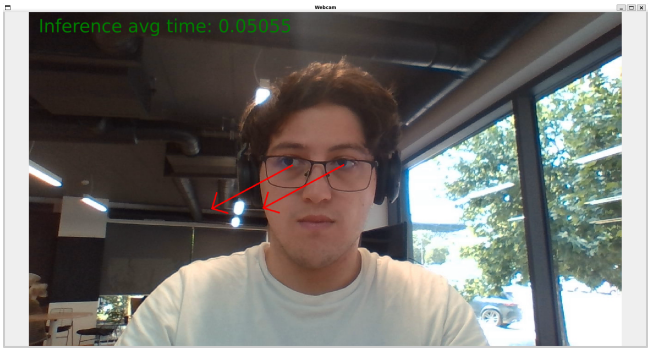


Figure 6.4: Enhanced real-time webcam. Gaze prediction is shown as 3D gaze vectors (red arrows). In the top left corner the average inference time is shown.

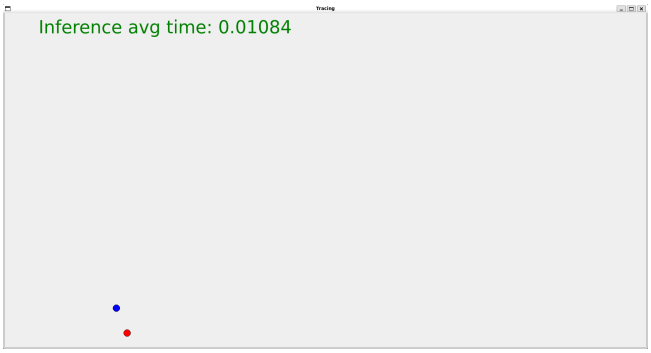


Figure 6.5: Tracing demo. A target (blue dot) is shown on top of the white canvas. The gaze prediction (red dot) and the average inference time are also shown.

Figure 6.6: GazeTrack graphical user interface

6.6 Performance analysis

At the end of the previous Chapter 5 I observed that my FastSightNet gaze estimation model takes about 0.013 seconds to process one frame (approximately 77

frames per second). My model's inference speed is beyond what I expect from a real-time system, however the model's input data is a face image which needs to be cropped out from the camera frame. In orderd to detect the face in the frame I am using RetinaFace inspired by the choice of Ahmed A. Abdelrahman et al. [4]. This model while having a great performance it is quite slow, it's inference speed being about 0.1 seconds per frame (10 frames per second), which slows down the system. Nevertheless, this face detection model is easily replaceable thanks to the system's property of low coupling.

System performance across diverse conditions

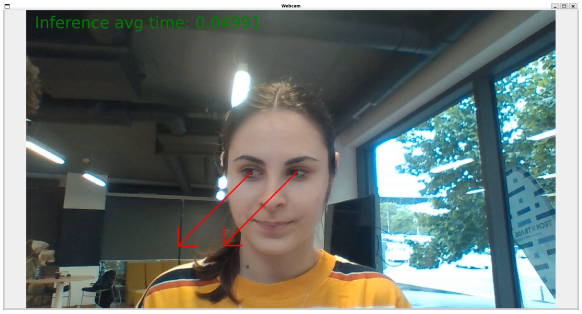
I have explored how the GazeTrack system performs on multiple subjects, different lightning conditions, and other commonly scenarios in the day-to-day life. First of all, I asked colleagues of mine to test the system. Because of ethical concerns, I collected the approval to use their face in my thesis project through a Google form. Results can be observed in Figure 6.7. It looks like the model generalizes well across other subjects.

I am aware that such a system should work well in different lightning conditions. In Figure 6.8 I showed the inference results for subjects placed in medium/low luminosity environments. Besides this, I also provide examples when the light source is placed at different angles relative to the subject. The model seems to work well in these scenarios as well. One reason can be the wide range of luminosity conditions in the training set. On top of that, eye-glasses reflections do not affect the performance as much as I thought it would.

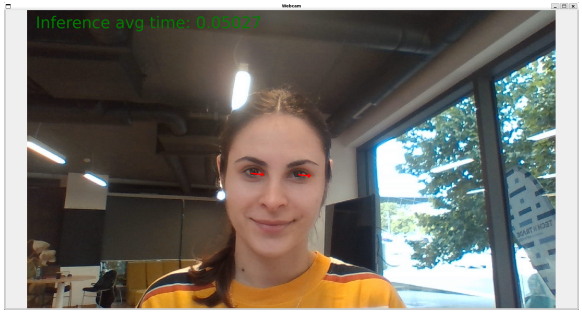
Lastly, I evaluated the model performance on subjects that are not centered (Figure 6.9c), frames that contain multiple subjects (Figure 6.9d), also blurry images (Figure 6.9e), and different head angles (Figure 6.9f). When multiple faces are detected in the frame, the gaze prediction is only performed on the first identified subject. An improvement to the system would be to perform gaze estimation on the subject with the biggest face area. The 3D gaze estimation works well even when the camera is not centered or the head is tilted, however the intersection with the 2D screen will surely be affected since I assumed the exact position of the subject relative to the camera.

I also did a visual estimation of the prediction error on the 2D demo, usually the prediction is between 1-7cm off, but it can also be 10 cm off in situation like looking at the bottom center of the screen. From my visual evaluation I find the average prediction error to be around 5cm. An improvement to the system would be to show the exact distance between the target and the prediction on screen. Nevertheless, in order for this system to work as an efficient Human-Computer Interface (e.g. to

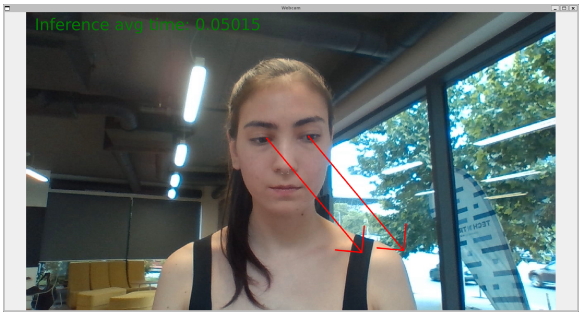
control cursor with the gaze) the prediction error should be further reduced.



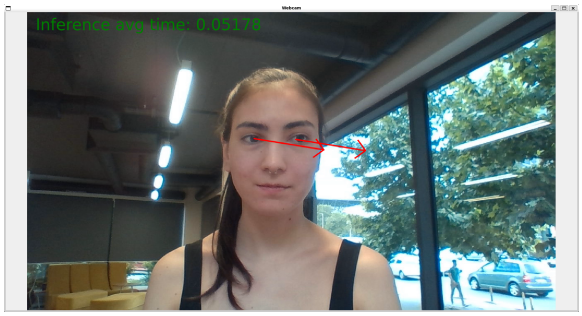
(a) Focusing buttom left



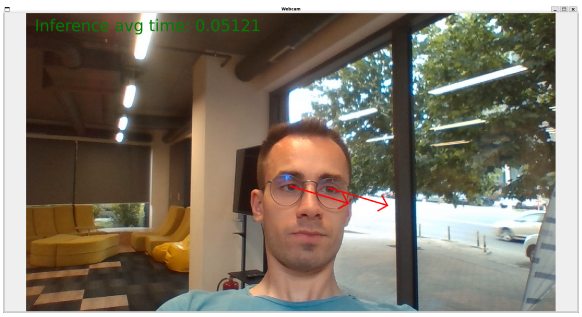
(b) Focusing the camera



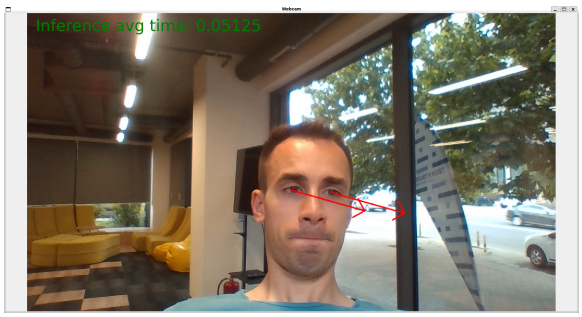
(c) Focusing bottom right



(d) Focusing top right

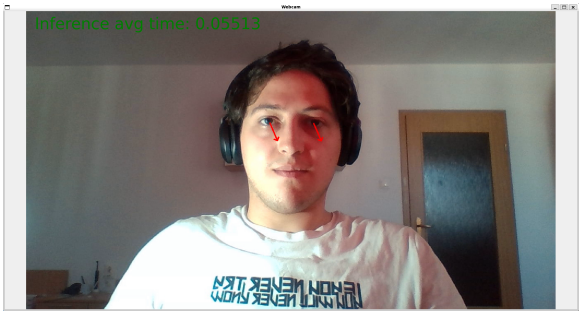


(e) Focusing top right (no glasses).

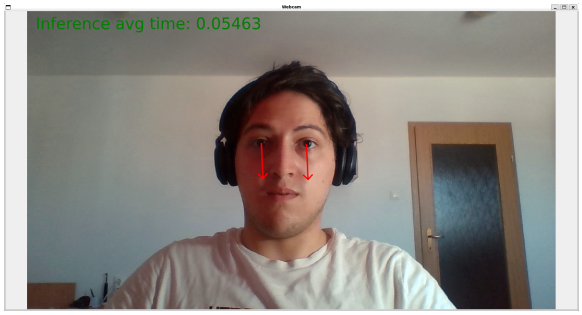


(f) Focusing top right (with eye-glasses).

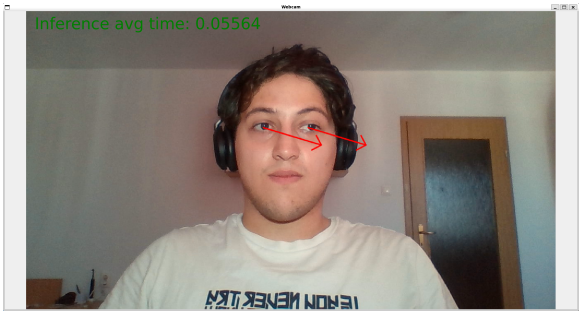
Figure 6.7: Different subjects testing the GazeTrack system.



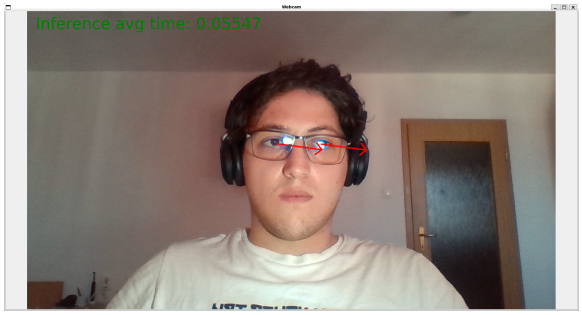
(a) Light source on the left



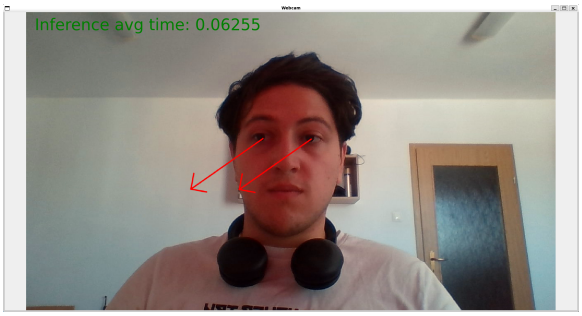
(b) Light source on the right



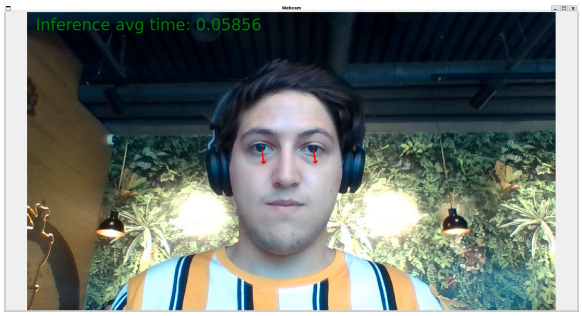
(c) Light source in the front



(d) Eye-glasses reflection

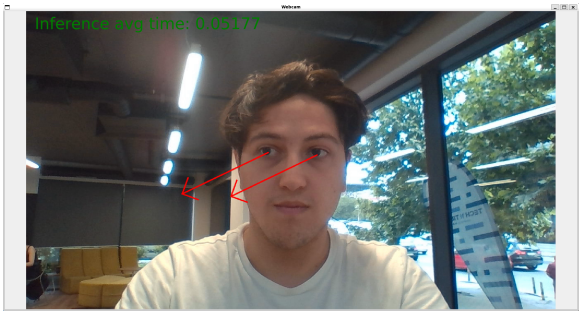


(e) Low luminosity

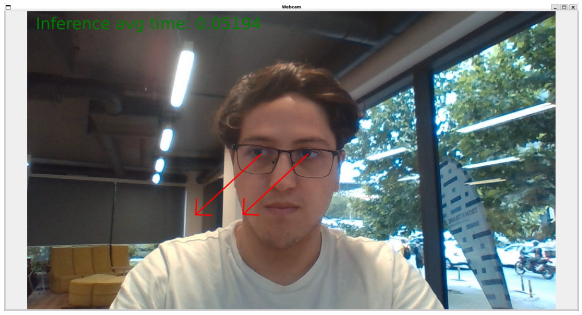


(f) Medium luminosity

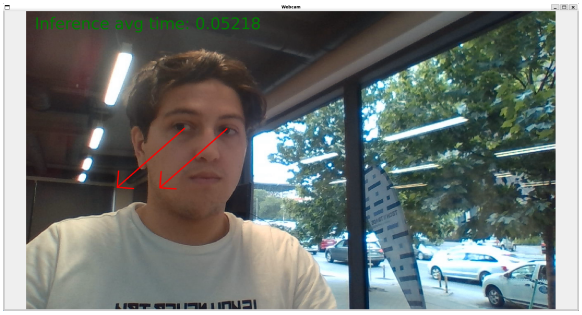
Figure 6.8: GazeTrack system tests in different lightining conditions.



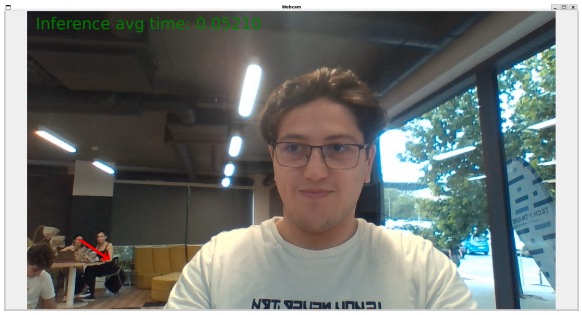
(a) Normal conditions



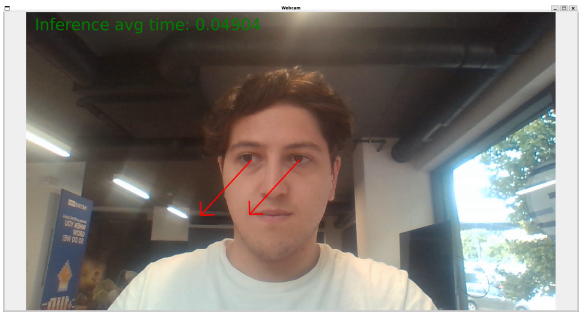
(b) With eye-glasses



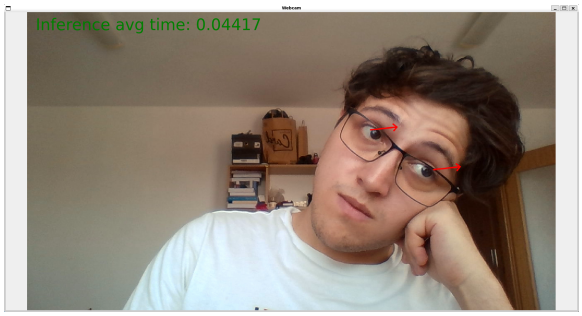
(c) Subject not centered



(d) Multiple faces



(e) Blurry image



(f) Tilted head pose

Figure 6.9: GazeTrack system tests in different environment and subject conditions.

Chapter 7

Usecase: Measuring Eye-Contact

Context

I have been involved in a research project at Babes-Bolyai University. The project was conducted in collaboration with our colleagues from the psychology faculty.

Idea

The idea behind the project was to analyze a conversation between two persons and predict how each one of them felt about the conversation (e.g. did they enjoy it or would they talk with that person again). A first step was to analyze each person from 4 perspective:

1. detect emotions from conversation transcripts
2. detect emotions from images with each subject
3. **detect periods of eye-contact**
4. detect smile or frowning from EMG sensors situated on the subject's face

Each method was developed by a separate team. The project overview can be observed in Figure 7.1. Second phase was to identify correlations between these pshycological responses and compute different metrics for each method. The project's final product will be an application that allows the visualization and comparison of different time series, and identifying correlations between these psychological responses.

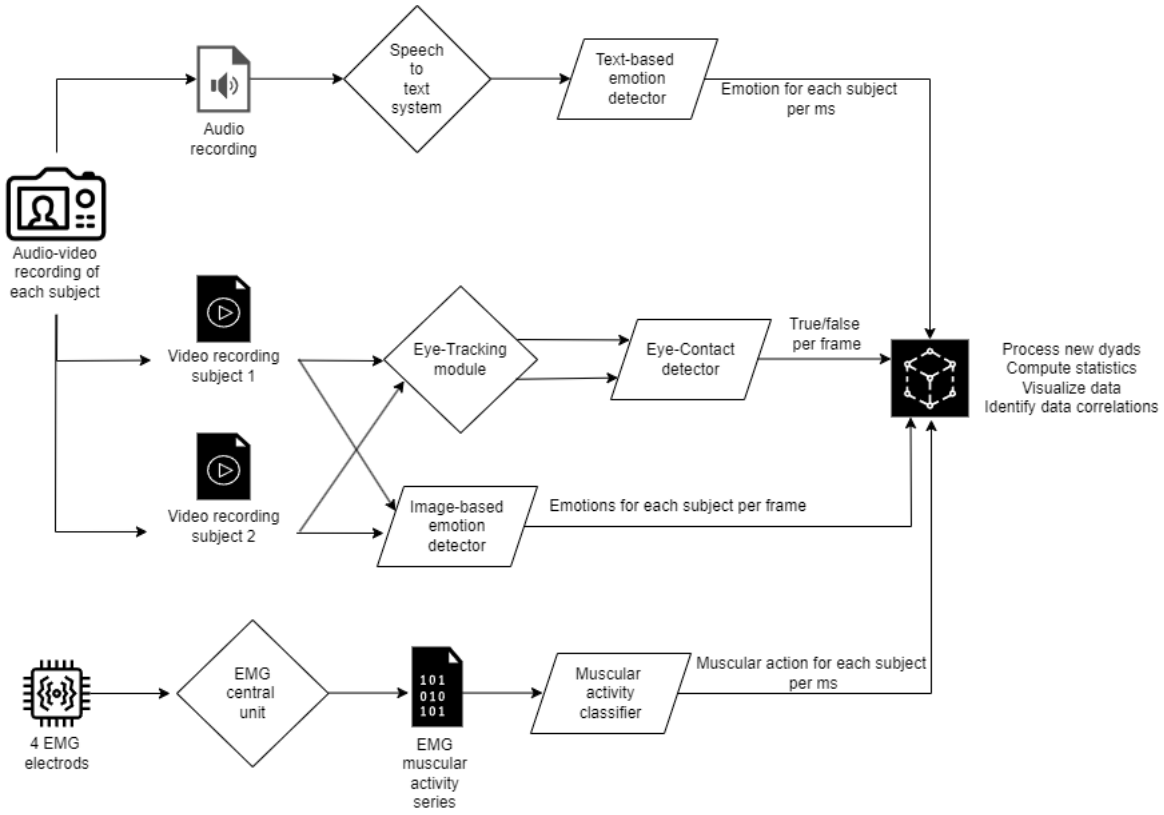


Figure 7.1: Research project overview

7.1 Methods

We decided to use a machine learning approach. Time was limited and we applied existing trained models on each task separately. In order to evaluate our solutions we collaborated with the psychology faculty to gather data.

Experiment setup

The experiment setup can be seen in Figure 7.2. Two subjects would sit at a table and talk about a subject of their choice. We set 2 cameras to record the head of each subject and also audio. On top of that, each person had 4 electrodes on their face for collecting EMG data.

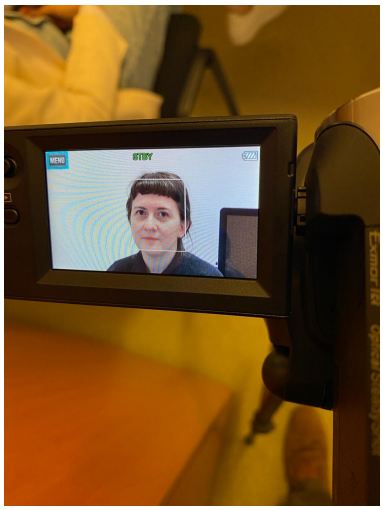
Data collection

For eye-contact detection we imposed a calibration period consisting of 3 phases:

1. eye-contact while standing still (30 sec)
2. eye-contact while moving (30 sec)
3. no eye-contact while moving (60 sec)

(a) Experiment room. Subjects will seat in front of each other on opposite sides of the table.

(b) Camera setup. The camera will be mounted on a tripod close to the right of the other subject.



(c) Camera view. The camera will capture the subject's head, allowing for slight movements.

Figure 7.2: Experiment setup

The EMG procedure also imposed a baseline period of 3 minutes, in which the subjects were not allowed to talk. The 10 minutes conversation followed. Each phase was delimited by a distinctive sound such that we could syncronize the recordings and cut out any mistake that would arise without stoping the cameras. The experiment was replicated with 12 dyads.

Measuring Eye-Contact

I was part of the team that measured Eye-Contact so I will elaborate on this part. To begin with, I splitted the problem in 2 phases. First phase is applying gaze estimation methods to each of the 2 subject and each frame. Secondly, use a classification model to decide if the person is looking at the other subject or not. A diagram of the Eye-Contact pipeline can be observed in Figure 7.3.

Gaze estimation

I decided to use a pretrained model for the gaze estimation task. The requirement for the model was that it must work offline and be open source. After exploring with multiple open source models, I decided to use the Intel OpenVINO gaze estimation model [3] since it works seamlessly on both single images and video files, while showing good performance. I saved the model output to a log file that I later processed.

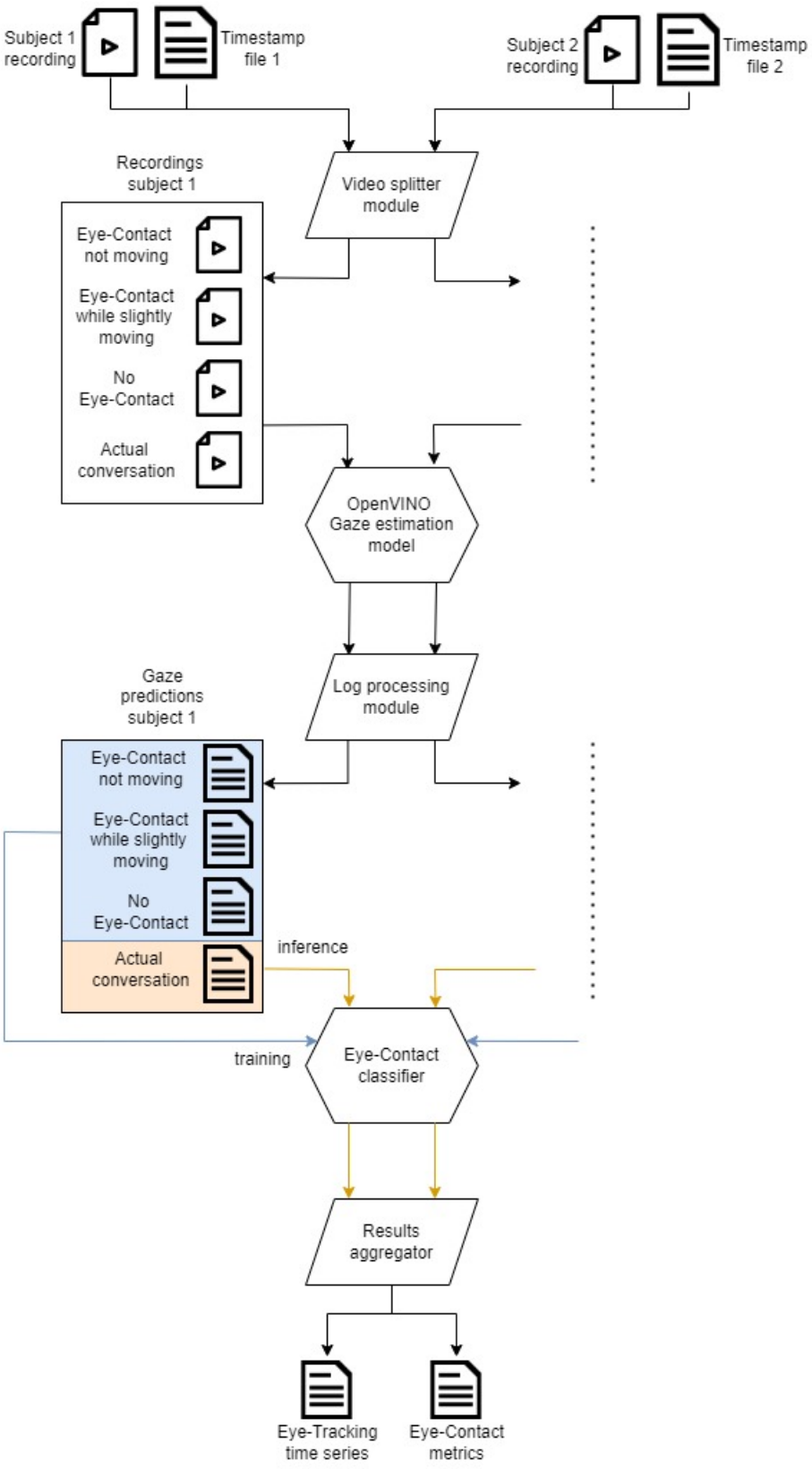


Figure 7.3: Eye-Contact module

Eye-Contact classifier

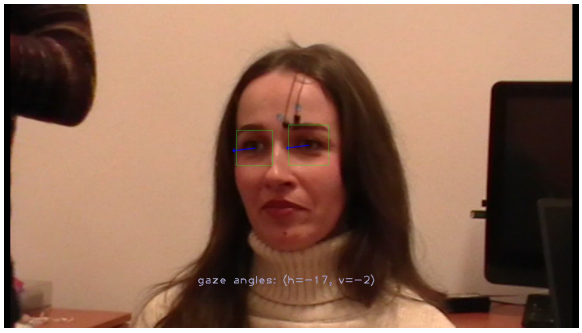
The Eye-Contact model consists of a K-Nearest Neighbors (KNN) classifier (with K equal to 10) which categorizes new data points into two classes: eye-contact or no eye-contact based on the closest neighbors in the input space.

7.2 Integrating FastSightNet

In order to showcase the results of the OpenVINO gaze estimation model [3] I have performed inference on a frame from one of the conversations we collected. The results can be seen in Figure 7.4a and Figure 7.4b. I also ran the FastSightNet model I proposed in Chapter 5 on these frames. As one can observe from Figure 7.4c and Figure 7.4d, the results are pretty similar.

One idea I have is to integrate the FastSightNet model into the Eye-Contact module for this project. This can be done quite seamlessly by replacing the model class that is being used. My model respects every requirement, since it is my own contribution and it works offline. One disadvantage of my model is the fact that it was not trained on subjects being more than 50 cm away from the camera, however it seems to generalize quite well to new scenarios.

(a) OpenVINO inference on subject 1



(b) OpenVINO inference on subject 2

(c) FastSightNet inference on subject 1

(d) FastSightNet inference on subject 2

Figure 7.4: Comparison between OpenVINO gaze estimation model and my FastSightNet evaluated on a single frame of the conversation. As far as privacy concerns go, the two subjects agreed to let me use the images in my thesis project by filling out a Google form.

Chapter 8

Conclusions

In this work, I approached the 3D appearance-based gaze estimation problem by first reproducing, training, tuning, and evaluating existing state-of-the-art models such as the L2CS-Net proposed by Ahmed A. Abdelrahman et al. [4]. Results were 18% worse compared to the original paper, with the reproduced model achieving 4.5° angular error on fold 2. I continued to call the reproduced model the baseline.

Next, I proposed a new model called FastSightNet based on the more efficient MobileNet architecture introduced by Andrew G. Howard et al. [19]. I performed hyperparameter tuning and compared the best model to the original L2CS-Net model. The FastSightNet model achieved 5.1° angular error on fold 2, however my model is 90% smaller in size and it can be trained 95% faster compared to the baseline, while at the same time the inference speed for the FastSightNet is 77 frames per second compared to the baseline speed of 24 frames per second.

In the next chapter, I introduced my GazeTrack system, which enables users to evaluate different models in both 3D and 2D scenarios. I closely followed the normal flow of system engineering by analyzing the requirements, designing my system, and testing it throughout the development process.

My proposed solution is not yet suitable for deployment due to the high prediction error of about 5 cm in 2D screen coordinates. However, I validated the idea that a MobileNet architecture is able to learn the complex patterns of the gaze estimation task.

SWOT analysis

Table 8.1 captures the main strengths and weaknesses of my approach, as well as the external opportunities that facilitated this work and possible threats to the idea of developing an intelligent human-computer interface based on eye-tracking technology.

Future work

I identified the limitations of the MPIIFaceGaze dataset I used in terms of gaze angle distribution (30° horizontally and 20° vertically), which motivates me to move towards other datasets such as the Gaze360 [23] dataset or the ETH-XGaze [52] dataset which has extreme gaze angles and might offer better results in unconstrained real-life scenarios.

An additional step forward would be to experiment with different layer configurations for my model or freeze fewer initial layers to help the model identify patterns in smaller sections of the input image. Additionally, I believe that using a higher-performance station will reduce the overall training time, enabling us to conduct more experiments.

Finally, I would add a new feature to the application that enables the movement of the cursor on the screen based on gaze predictions. The next step would be to build a software that performs gaze estimation in the background and moves the cursor seamlessly in the desired position on the screen. This will allow the user to navigate across other applications freely by only using eye-tracking, fully replacing mouse or trackpad input.

Strengths	1. No special hardware is required, this solution should work on most laptops with a webcam. 2. The inference works in real-time while the training is faster, compared to other approaches that are more computationally intensive. 3. My model performs 3D gaze estimation very well in practice (e.g. different backgrounds, subjects, or lightning conditions).
Weaknesses	1. The dataset I trained on may not be well suited for daily use-cases, because of limited gaze distribution. 2. Being an appearance-based gaze estimator, my solution may not work well in unseen environmental conditions. 3. In order for my solution to be deployed in real-life scenarios, it requires a face detection model beforehand, which may add more delay.
Opportunities	1. My solution may be one of the next human-computer interfaces, as well as one of the most intuitive out there (e.g. compared to the mouse and touchpad). 2. People with disabilities can benefit from my solution by enabling them to control their computers. 3. The latest advancements in deep learning, including convolutional neural networks and visual transformers, provide new ways of approaching the gaze estimation task. 4. Hardware performance got a lot better over the years, which enables me to perform more experiments in a shorter amount of time.
Threats	1. The fact that my software uses the camera all the time for predicting the gaze, prevents other applications from using it in the meantime, which would determine the user to deactivate the system in some situations (e.g. meetings). 2. New state-of-the-art models may obtain better performance, which would make my model obsolete, and therefore others may capture the majority of the market (e.g. big tech companies). 3. AI regulations as well as GDPR rules may not allow this solution to be released in production, because it processes personal data.

Table 8.1: SWOT Analysis

Bibliography

[1] Gazehub. https://phi-ai.buaa.edu.cn/Gazehub/. Accessed: 2024-04-23.

[2] Weights & biases: Developer tools for machine learning. https://wandb.ai/site. Accessed: 2024-06-09.

[3] Gaze estimation demo. https://docs.openvino.ai/2024/omz_demos_gaze_estimation_demo_cpp.html, 2024. Accessed: 2024-06-09.

[4] Ahmed A. Abdelrahman, Thorsten Hempel, Aly Khalifa, and Ayoub Al-Hamadi. L2cs-net: Fine-grained gaze estimation in unconstrained environments. *CoRR*, abs/2203.03339, 2022.

[5] Abien Fred Agarap. Deep learning using rectified linear units (relu). *CoRR*, abs/1803.08375, 2018.

[6] A. Anusha and Syed Ahmed. Vehicle tracking and monitoring system to enhance the safety and security driving using iot. pages 49–53, 07 2017.

[7] Rishi Athavale, Lakshmi Sritan Motati, and Rohan Kalahasty. One eye is all you need: Lightweight ensembles for gaze estimation with single encoders. *CoRR*, abs/2211.11936, 2022.

[8] Pietro Bonazzi, Thomas Rüegg, Sizhen Bian, Yawei Li, and Michele Magno. Tinytracker: Ultra-fast and ultra-low-power edge vision in-sensor for gaze estimation. In *2023 IEEE SENSORS, Vienna, Austria, October 29 - Nov. 1, 2023*, pages 1–4. IEEE, 2023.

[9] Radu Bozomitu, Vlad Cehan, Robert Lupu, Cristi Rotariu, and Constantin Barabasa. A new technique for improving pupil detection algorithm. pages 1–4, 07 2015.

[10] Yihua Cheng, Haofei Wang, Yiwei Bao, and Feng Lu. Appearance-based gaze estimation with deep learning: A review and benchmark. *CoRR*, abs/2104.12668, 2021.

[11] PyTorch Contributors. Pytorch. `https://pytorch.org/`, 2024. Accessed: 2024-06-09.

[12] Tobias Fischer, Hyung Jin Chang, and Yiannis Demiris. RT-GENE: real-time eye gaze estimation in natural environments. In Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss, editors, *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part X*, volume 11214 of *Lecture Notes in Computer Science*, pages 339–357. Springer, 2018.

[13] Hossein Gholamalinezhad and Hossein Khosravi. Pooling methods in deep neural networks, a review. *CoRR*, abs/2009.07485, 2020.

[14] Steven Gong. How does a neural network work intuitively in code. `https://blog.stevengong.co/how-does-a-neural-network-work-intuitively-in-code/f51f7b2c1e3f?gi=537f3f66f14b`, 2023. Accessed: 2024-06-09.

[15] Alex Graves, Santiago Fernández, and Jürgen Schmidhuber. Bidirectional LSTM networks for improved phoneme classification and recognition. In Wlodzislaw Duch, Janusz Kacprzyk, Erkki Oja, and Slawomir Zadrozny, editors, *Artificial Neural Networks: Formal Models and Their Applications - ICANN 2005, 15th International Conference, Warsaw, Poland, September 11-15, 2005, Proceedings, Part II*, volume 3697 of *Lecture Notes in Computer Science*, pages 799–804. Springer, 2005.

[16] Yiran Guan, Zhuoguang Chen, Wenzheng Zeng, Zhiguo Cao, and Yang Xiao. End-to-end video gaze estimation via capturing head-face-eye spatial-temporal interaction context. *IEEE Signal Process. Lett.*, 30:1687–1691, 2023.

[17] Susumu Harada, James A. Landay, Jonathan Malkin, Xiao Li, and Jeff A. Bilmes. The vocal joystick: : evaluation of voice-based cursor control techniques. In Simeon Keates and Simon Harper, editors, *Proceedings of the 8th International ACM SIGACCESS Conference on Computers and Accessibility, ASSETS 2006, Portland, Oregon, USA, October 23-25, 2006*, pages 197–204. ACM, 2006.

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[19] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017.

[20] Bulat Ibragimov and Claudia Mello-Thoms. The use of machine learning in eye tracking studies in medical imaging: A review. *IEEE journal of biomedical and health informatics*, PP, 02 2024.

[21] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Francis R. Bach and David M. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 448–456. JMLR.org, 2015.

[22] Jürgen Keller, Amon Krimly, Lisa Bauer, Sarah Schulenburg, Sarah Böhm, Helena Aho-Özhan, Ingo Uttner, Martin Gorges, Jan Kassubek, Elmar Pinkhardt, Sharon Abrahams, Albert Ludolph, and Dorothée Lulé. A first approach to a neuropsychological screening tool using eye-tracking for bedside cognitive testing based on the edinburgh cognitive and behavioural als screen. *Amyotrophic Lateral Sclerosis and Frontotemporal Degeneration*, 18:1–8, 04 2017.

[23] Petr Kellnhofer, Adrià Recasens, Simon Stent, Wojciech Matusik, and Antonio Torralba. Gaze360: Physically unconstrained gaze estimation in the wild. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, pages 6911–6920. IEEE, 2019.

[24] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[25] Ahmad F. Klaib, Nawaf O. Alsrehin, Wasen Y. Melhem, Haneen O. Bashtawi, and Aws Abed Al Raheem Magableh. Eye tracking algorithms, techniques, tools, and applications with an emphasis on machine learning and internet of things technologies. *Expert Syst. Appl.*, 166:114037, 2021.

[26] Rakshit Sunil Kothari, Shalini De Mello, Umar Iqbal, Wonmin Byeon, Seonwook Park, and Jan Kautz. Weakly-supervised physically unconstrained gaze estimation. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021, virtual, June 19-25, 2021*, pages 9980–9989. Computer Vision Foundation / IEEE, 2021.

[27] Kyle Krafka, Aditya Khosla, Petr Kellnhofer, Harini Kannan, Suchendra M. Bhandarkar, Wojciech Matusik, and Antonio Torralba. Eye tracking for everyone. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 2176–2184. IEEE Computer Society, 2016.

[28] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In Peter L. Bartlett, Fernando C. N. Pereira, Christopher J. C. Burges, Léon Bottou, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*, pages 1106–1114, 2012.

[29] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proc. IEEE*, 86(11):2278–2324, 1998.

[30] Itziar Lozano, Ruth Campos, and Mercedes Belinchón. Eye-tracking measures in audiovisual stimuli in infants at high genetic risk for ASD: challenging issues. In Bonita Sharif and Krzysztof Krejtz, editors, *Proceedings of the 2018 ACM Symposium on Eye Tracking Research & Applications, ETRA 2018, Warsaw, Poland, June 14-17, 2018*, pages 73:1–73:3. ACM, 2018.

[31] The Qt Company Ltd. Qt for python. `https://doc.qt.io/qtforpython-6/`, 2024. Accessed: 2024-06-09.

[32] Mostafa Mohammadi, Hendrik Knoche, Michael Gaihede, Bo Bentsen, and Lotte N. S. Andreasen Struijk. A high-resolution tongue-based joystick to enable robot control for individuals with severe disabilities. In *16th IEEE International Conference on Rehabilitation Robotics, ICORR 2019, Toronto, ON, Canada, June 24-28, 2019*, pages 1043–1048. IEEE, 2019.

[33] Pallavi Mohan, Wooi Boon Goh, Chi-Wing Fu, and Sai-Kit Yeung. Dualgaze: Addressing the midas touch problem in gaze mediated VR interaction. In *IEEE International Symposium on Mixed and Augmented Reality, ISMAR 2018 Adjunct, Munich, Germany, October 16-20, 2018*, pages 79–84. IEEE, 2018.

[34] Kenneth Alberto Funes Mora, Florent Monay, and Jean-Marc Odobez. EYEDIAP: a database for the development and evaluation of gaze estimation algorithms from RGB and RGB-D cameras. In Pernilla Qvarfordt and Dan Witzner Hansen, editors, *Eye Tracking Research and Applications, ETRA '14, Safety Harbor, FL, USA, March 26-28, 2014*, pages 255–258. ACM, 2014.

[35] Zaid Bin Muzammil. Unleashing the power of mobilenet: A comparison with simple convolutional neural networks. `https://medium.com/@zaidbinmuzammil123/unleashing-the-power-of-mobilenet/a-comparison-with-simple-convolutional-neural-networks/71d49f8c86ef`, 2023. Accessed: 2024-06-09.

[36] Luis F. Nicolás-Alonso and Jaime Gómez Gil. Brain computer interfaces, a review. *Sensors*, 12(2):1211–1279, 2012.

[37] Toni Pakkanen and Roope Raisamo. Appropriateness of foot interaction for non-accurate spatial tasks. In Elizabeth Dykstra-Erickson and Manfred Tscheligi, editors, *Extended abstracts of the 2004 Conference on Human Factors in Computing Systems, CHI 2004, Vienna, Austria, April 24 - 29, 2004*, pages 1123–1126. ACM, 2004.

[38] Cristina Palmero, Javier Selva, Mohammad Ali Bagheri, and Sergio Escalera. Recurrent CNN for 3d gaze estimation using appearance and shape cues. In *British Machine Vision Conference 2018, BMVC 2018, Newcastle, UK, September 3-6, 2018*, page 251. BMVA Press, 2018.

[39] Se Jin Park, Murali Subramaniyam, Seunghee Hong, Damee Kim, and Jaehak Yu. Conceptual design of the elderly healthcare services in-vehicle using iot. 03 2017.

[40] Seonwook Park, Shalini De Mello, Pavlo Molchanov, Umar Iqbal, Otmar Hilliges, and Jan Kautz. Few-shot adaptive gaze estimation. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, pages 9367–9376. IEEE, 2019.

[41] David E. Rumelhart and James L. McClelland. *Learning Internal Representations by Error Propagation*, pages 318–362. 1987.

[42] Rustam Shadiev and Dandan Li. A review study on eye-tracking technology usage in immersive virtual reality learning environments. *Comput. Educ.*, 196:104681, 2023.

[43] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[44] Yusuke Sugano, Yasuyuki Matsushita, and Yoichi Sato. Learning-by-synthesis for appearance-based 3d gaze estimation. In *2014 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2014, Columbus, OH, USA, June 23-28, 2014*, pages 1821–1828. IEEE Computer Society, 2014.

[45] Yuyang Sun, Qingzhong Li, Honggen Zhang, and Jiancheng Zou. The application of eye tracking in education. In Jeng-Shyang Pan, Pei-Wei Tsai, Junzo Watada, and Lakhmi C. Jain, editors, *Advances in Intelligent Information Hiding*

and Multimedia Signal Processing - Proceedings of the Thirteenth International Conference on Intelligent Information Hiding and Multimedia Signal Processing, IIH-MSP 2017, August, 12-15, 2017, Matsue, Shimane, Japan, Part II, volume 82 of *Smart Innovation, Systems and Technologies*, pages 27–33. Springer, 2017.

[46] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.

[47] OpenCV Team. Opencv - open computer vision library. `https://opencv.org/`, 2024. Accessed: 2024-06-09.

[48] Eric Whitmire, Laura C. Trutoiu, Robert Cavin, David Perek, Brian Scally, James Phillips, and Shwetak N. Patel. Eyecontact: scleral coil eye tracking for virtual reality. In Michael Beigl, Paul Lukowicz, Ulf Blanke, Kai Kunze, and Seungyon Claire Lee, editors, *Proceedings of the 2016 ACM International Symposium on Wearable Computers, ISWC 2016, Heidelberg, Germany, September 12-16, 2016*, pages 184–191. ACM, 2016.

[49] Wikipedia contributors. Human–computer interaction. `https://en.wikipedia.org/wiki/Human%E2%80%93computer_interaction`. Accessed: May 11, 2024.

[50] Saining Xie, Ross B. Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 5987–5995. IEEE Computer Society, 2017.

[51] Begoña García Zapirain, Isabel de la Torre Díez, and Miguel López Coronado. Dual system for enhancing cognitive abilities of children with ADHD using leap motion and eye-tracking technologies. *J. Medical Syst.*, 41(7):111:1–111:8, 2017.

[52] 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. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part V*, volume 12350 of *Lecture Notes in Computer Science*, pages 365–381. Springer, 2020.

[53] Xucong Zhang, Yusuke Sugano, Mario Fritz, and Andreas Bulling. Appearance-based gaze estimation in the wild. In *IEEE Conference on Computer*

Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015, pages 4511–4520. IEEE Computer Society, 2015.

[54] Xucong Zhang, Yusuke Sugano, Mario Fritz, and Andreas Bulling. It's written all over your face: Full-face appearance-based gaze estimation. In *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 2299–2308. IEEE Computer Society, 2017.

[55] Xucong Zhang, Yusuke Sugano, Mario Fritz, and Andreas Bulling. Mpiigaze: Real-world dataset and deep appearance-based gaze estimation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 41(1):162–175, 2019.