Technical University of Crete
School of Electrical and Computer Engineering

Diploma Thesis

A Gaze Prediction Model for Task-Oriented VR Environments

Konstantina Mammou

Thesis Committee
Professor Aikaterini Mania (supervisor)
Professor Michail Zervakis
Asst. Professor Nikos Giatrakos

Chania | Crete, December 2024

Abstract

Gaze prediction in Virtual Reality (VR) has attracted significant attention due to its potential to enhance user interaction and optimize VR applications, such as gaze-contingent rendering. The dynamic and immersive nature of VR environments presents unique challenges, especially in predicting gaze in task-oriented environments compared to free-viewing or static ones. This thesis proposes a model for predicting gaze in such environments, investigating the role and ability of temporal continuity to enable accurate predictions.

The proposed model is composed of three key modules. The Image Sequence Module (ISM) utilizes ConvLSTM layers to capture temporal motion features from sequences of frames, while the Gaze Sequence Module (GSM) employs LSTM layers to extract temporal patterns from gaze data. These outputs are combined in the Fusion Module, which integrates information from both ISM and GSM to predict a single gaze point. The OpenNEEDS dataset, offering diverse VR scenarios and gaze recordings, was used for training. Preprocessing steps included frame and gaze point normalization, conversion of 3D gaze vectors to 2D visual angles, outlier removal, and sequence creation to prepare the data for the model.

The model was evaluated with metrics such as angular error and recall rate, with the model significantly outperforming baseline methods. However, the runtime performance remains a limitation, indicating the need for optimization for real-time applications. Our work contributes a robust, adaptable, consistent model for gaze prediction in task-oriented VR environments and demonstrates the potential of leveraging temporal continuity for accurate gaze prediction.

Acknowledgements

I would like to initially thank my supervisor, Professor Katerina Mania, for her guidance and support throughout the development of this thesis.
Next, a shoutout to the members of the Surreal team for their assistance, especially during the early stages of this project.
I am deeply thankful to my partner Michalis and my friends Anna, Giorgos, and Chris, who have helped me in more ways than I can count over the years.
To my childhood friends Antonia, Dimitra, and Spyros, thank you for always being there for me.
Lastly, I am truly grateful to my family for their infinite support and belief in me.

Contents

List of Figures

LIST OF FIGURES

Chapter 1

Introduction

In this chapter, the motivation behind our work is presented. A brief description of the implementation, results, and contributions is provided, along with an outline of the structure of the entire thesis.

1.1 Motivation

Gaze prediction is a complex subject that has recently seen significant improvements with the use of deep neural networks. Understanding and predicting where users look is not only fascinating in terms of studying human behaviour but also has numerous practical applications. In fields such as human-computer interaction (HCI), advertising, accessibility, gaze prediction can enhance user interaction and experience. Moreover, leveraging gaze prediction, image and video compression techniques as well as rendering algorithms can be significantly optimized, thereby improving performance.

Concurrently, Virtual Reality (VR) has undergone rapid growth over the past decade. The introduction of affordable head-mounted displays (HMDs) has increased the popularity of VR devices, with more and more people exploring the field and the experiences it offers. VR's potential to provide immersive and interactive experiences has attracted considerable research interest, with applications ranging from gaming and education to healthcare and art.

In the context of VR, users' gaze information becomes increasingly important, as VR presents new areas that can utilize said information, such as VR content design,

eye-movement based interactions, gaze-contingent rendering, etc. Improving the performance of HMDs is essential, as it enables more natural and immersive interactions. Currently, the most common solution is based on hardware-based eye-trackers. However, eye-trackers can be expensive and can present new issues that need to be solved, such as latency. Therefore, gaze estimation in VR has slowly become an active area of study.

Predicting gaze in VR environments presents unique challenges compared to desktop environments. The dynamic nature of VR content, coupled with the freedom of movement and the different point of view provided by HMDs, results in distinct gaze behaviour patterns. Although existing models for static images and desktop environments have shown promising results, gaze prediction models for VR and dynamic scenes are still of early stages. There are many limitations due to the complexity of a virtual environment, especially when there is a task involved. Recently, the concept of temporal continuity of visual attention has been presented, and researchers are trying to determine how it can be exploited for more accurate predictions in complex environments.

Our research highlighted key limitations in existing gaze prediction models, including the need for task-specific focus in VR, generalized adaptability across tasks, simplified input requirements for integration, and lightweight architectures capable of rapid predictions. This led us to explore the relationship between task-specific environments and temporal continuity as the foundation for a more accurate and flexible model.

1.2 Brief Description

In this thesis, we developed a gaze prediction model for task-oriented VR environments based on deep neural networks. Our focus lies on dynamic environments, where users actively perform tasks. Based on sequences of past gaze points, as well as past frames, we investigate the role of temporal continuity in prediction models. We analyze human gaze behaviour in VR environments, based on public datasets that contain recorded gaze points from users interacting with virtual environments.

More specifically, the model consists of three modules: the *image sequence module (ISM)*, which learns the temporal motion features among consecutive frames; the *gaze sequence module (GSM)*, which captures the temporal patterns of user gaze; and the *fusion module (FM)*, which combines the outcomes from the first two modules to make a single gaze prediction.

The ISM is designed under the assumption that some knowledge of the scene is crucial for identifying stimuli or salient areas in the frame that naturally draw attention. ConvLSTM layers allow us to process a sequence of frames and memorize temporal information, learning correlations across frames. These layers combine the strong long-term memory capabilities of LSTM layers with the ability of convolutional layers to capture image features. This module consists of 5 ConvLSTM2D layers with ReLU as their activation function, 4 MaxPooling layers to reduce dimensionality, and 4 fully connected (FC) layers to encode the learned information, with sizes of 64, 32, 32, and 16 respectively. A dropout layer (dropout rate = 0.5) is also included to prevent overfitting. The input to this module is a sequence of 10 continuous frames.

The GSM, on the other hand, captures the temporal patterns of user gaze. This module is based on LSTM layers, which are well-suited for long-term memory. It consists of 4 LSTM layers with ReLU as the activation function, 2 FC layers with sizes of 32 and 16 respectively, and a dropout layer (dropout rate = 0.5). The input to this module is a sequence of 10 continuous gaze points.

Finally, the FM merges the outputs from the ISM and GSM using a Maximum operation. This fusion process integrates both the temporal frame information and past gaze point data. Following the merge, a fully connected layer of size 16 and ReLU as the activation function is employed to further process the combined features, as well as a dropout layer to prevent overfitting (dropout rate = 0.5). The final output layer consists of a fully connected layer of size 2, with sigmoid as the activation function. This layer generates the final gaze prediction as a two-dimensional output (x_g, y_g), with values normalized between 0 and 1.

The OpenNEEDS dataset, a newly published large-scale, open-source dataset, was used for this work. It contains data from 44 users who freely explored virtual environments and performed various tasks. The dataset includes two different types of scenes (indoor and outdoor) and numerous tasks, such as reading, throwing, object-manipulation, drawing, aiming, and shooting. For this study, scene information (frames at 8-bit resolution in sRGB color space, down-sampled to a pixel resolution of 128×71) and gaze information (3D gaze vectors) were used. For each sample, there was additional information necessary such as a user ID, a scene ID, and a timestamp.

The dataset was preprocessed to fit the model's requirements. Frames were normalized to values between 0 and 1, and gaze points were converted from 3D vectors to 2D

visual angles, which were also normalized between 0 and 1. Outliers were removed using the Interquartile Range Method to improve model robustness and performance. Input sequences were then created, with each sequence consisting of consecutive frames and their corresponding gaze points. The label for each sequence corresponds to the gaze point of the next frame, ensuring that the model learns to predict the future gaze point based on the current sequence of frames and gaze information, without needing the next frame. Python generators were implemented to handle the data in manageable batches, as loading all the images at once would overwhelm the available RAM. Each potential sequence was verified to ensure that all frames originated from the same user and scene. Separate generators were implemented for the training, validation, and testing datasets.

For training the model, Mean Absolute Error (MAE) was used as the loss function, and the Adam optimizer was employed with an initial learning rate of 0.001. The training process used a batch size of 64 and was conducted over 10 epochs. Two callback mechanisms—Reduce LR On Plateau and Early Stopping—were used to improve the training process. The model was trained on Google Colab using the NVIDIA L4 Tensor Core GPU.

Figure 1.1: Examples of our results. The purple (+) cross denotes the ground truth gaze position, with the purple circle illustrating the foveal region with radius 15°. The yellow (+) cross represents the prediction of our model, the red (+) cross shows the center baseline and the blue (+) the mean baseline. We can see that the model performs better compared to the baselines.

The model was evaluated based on its prediction error (measured in degrees), recall rate (for assessing potential use in foveated rendering pipelines), and runtime performance. Two baselines (center and mean) were defined for comparison. The angular distance between the predicted gaze position and the ground truth was used to evaluate

the model's performance. The results are promising. The model achieved a median error of approximately 4 degrees with a small interquartile range (IQR), indicating robustness, accuracy, and consistency. In comparison, the center baseline had a median of 13.15 degrees with a large IQR, and the mean baseline had a median of 11.95 degrees with a similarly large IQR. Additionally, the model achieved a recall rate of 99.87%, demonstrating its practical applicability for replacing or assisting an eye tracker. In contrast, the center and mean baselines had recall rates of 60.8% and 69.46%, respectively. However, the average runtime performance remains a significant limitation. The model's complex architecture, while accurate, is computationally intensive, and optimization is needed to make the model more lightweight and faster, even if it requires sacrificing some accuracy for processing speed.

1.3 Contributions

The main contributions of this thesis are:

- A deep exploration, both theoretical and practical, of the concept of temporal continuity for gaze prediction in VR.
- The development of a model that achieves strong accuracy and consistency with data from various tasks and users, demonstrating its adaptability and generalizability.
- The creation of a model that requires simple inputs, making it easy to integrate into various systems.

1.4 Structure of the Thesis

- Chapter 1: Introduction

  This chapter introduces the concepts of gaze prediction and Virtual Reality. It also provides a summary of our implementation, results, and outlines the structure of the entire thesis.

- Chapter 2: Research Overview

  This chapter presents an overview of the research conducted for this thesis, covering the key concepts that inspired and guided the work.

- Chapter 3: Theoretical and Technological Background

  This chapter provides the necessary background for this thesis, including theoretical concepts such as neural networks and their operation, as well as the software tools used in the implementation.

- Chapter 4: Implementation

  This chapter provides a detailed explanation of the system implementation. It covers the process from dataset selection and preprocessing to the design, implementation, and training of our model.

- Chapter 5: Evaluation

  This chapter presents the evaluation of our system, defining the evaluation metrics and discussing the results in detail.

- Chapter 6: Conclusions & Future Work

  The final chapter summarizes the conclusions drawn from the implementation of this thesis. It also provides potential future avenues for further improvement of the results.

Chapter 2

Research Overview

In this chapter we will elaborate on the necessary research conducted for this thesis. We will begin by exploring the concept of Virtual Reality, the use of head-mounted displays, and the technique of foveated rendering. Next, we will examine the anatomy and function of the human eye, followed by an in-depth analysis of gaze behaviour. Finally, we will discuss the field of gaze prediction, highlighting state-of-the-art models and their limitations, which ultimately inspired the development of our approach.

2.1 Virtual Reality

Virtual Reality (VR) is a technology that has been envisioned for decades, evolving from conceptual ideas to sophisticated systems that offer immersive experiences. This section provides a brief history of VR, highlighting key developments and milestones.

2.1.1 Brief History of VR

2.1.1.1 Early Concepts and Prototypes

The concept of VR can be traced back to the mid-20th century. Cinematographer Morton Heilig created the Sensorama [25], the first VR machine (patented in 1962). It was a large booth that could accommodate up to four people simultaneously. The Sensorama combined multiple technologies to stimulate all of the senses: it featured full colour 3D video, audio, vibrations, smell and atmospheric effects, such as wind. Heilig envisioned Sensorama as the "cinema of the future" and aimed to fully immerse people in their films.

Figure 2.1: Sensorama Prototype [1]

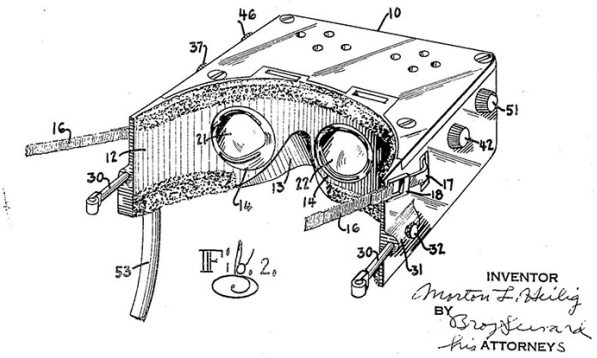


Figure 2.2: A Drawing from Telesphere Mask Patent [2]

2.1.1.2 The Birth of Head-Mounted Displays

Morton Heilig's next invention was the Telesphere Mask (patented in 1960), the first example of a head-mounted display (HMD), albeit for the non-interactive film medium without any motion tracking. The headset provided stereoscopic 3D and wide vision with stereo sound. In 1968 Ivan Sutherland and his student Bob Sproull created the first VR / AR head mounted display, the "Sword of Damocles", which was connected to a computer and not a camera [26].

Figure 2.3: Sword Of Damocles [2]

2.1.1.3 Advancements in the 1980s and 1990s

In 1975, Krueger's VIDEOPLACE, the first interactive VR platform, was presented. It used computer graphics, projectors, video cameras, video displays and position-sensing technology to display computer-generated silhouettes of users, mimicking their movements.

In 1985, the first company to sell VR goggles and gloves was founded, VPL Research, Inc. They introduced several key innovations, including the DataGlove and the EyePhone HMD [27]. The military used VR extensively, from flight simulation to exposure therapy for the treatment of PTSD in war veterans. NASA also invested in VR technologies to train astronauts [28].

Figure 2.4: A demonstration of the EyePhone system, which uses special goggles, and a DataGlove, allowing users to see and manipulate objects around in a computer created environment [3].

2.1.1.4 The Rise of Consumer VR

In 2012, Palmer Lucky launched a Kickstarter campaign to fund the product and development of his prototype headset, the Oculus Rift. The campaign raised almost 2.5 million dollars and was a clear dividing line between the commercial failures of consumer VR in the past and the modern VR revolution. The acquisition of Oculus VR company by Facebook, along with the releases of VR

Figure 2.5: Oculus Rift CV1 [4]

headsets by several major technology companies such as Google, Samsung, HTC and Sony, led to radical advancements in computing and display technologies. These developments made VR more accessible to consumers and developers.

2.1.2 Head Mounted Displays

A head-mounted display (HMD) is a device worn on the head that places a display screen directly in front of the user's eyes. These displays immerse the user in a digital environment by presenting stereoscopic images for each eye, creating a sense of depth and a 3D visual experience. HMDs vary in design, from lightweight and compact models for casual use to more sophisticated systems designed for professional applications.

HMDs are primarily used to enable immersive experiences, providing users with the ability to interact with digital environments in ways that feel physically present. Some key areas where HMDs are widely employed include gaming, education, training simulations (medical, military, aviation, F1). They can also be found in industrial application for real-time guidance or in fields like architecture and design. Finally, a very important area is healthcare, where HMDs can be used for patient rehabilitation and mental health treatments, such as exposure therapy for anxiety disorders.

The most popular commercially available HMDs currently are the Valve Index, Meta Quest 3 and the HTC Vive Pro Eye, each one with its strengths and weaknesses.

Figure 2.6: Valve Index [5]

Known for its high refresh rate and precise tracking, the Valve Index is favoured by PC VR enthusiasts. It offers top-tier visual fidelity and features like finger-tracking controllers, making it suitable for advanced gaming and interactive experiences.

Figure 2.7: Meta Quest 3 [6]

One of the most popular standalone VR headsets, The Meta Quest 3 offers high-quality graphics, thinner form factor and lenses, as well as additional sensors and colour passthrough cameras intended for mixed reality (MR) software.

Figure 2.8: HTC Vive Pro Eye [7]

Finally, the HTC Vive Pro Eye headset includes built-in Tobii eye-tracking technology, making it a popular choice for gaze-based research. Its real-time eye-tracking capabilities allow for detailed analysis of where users are looking, how long their gaze stays on certain objects, and how attention shifts during VR experiences, making it ideal for applications in research, training, and behavioural studies.

2.1.3 Immersion

Immersion in Virtual Reality refers to the degree to which a user feels present in a virtual environment. This is primarily achieved through sensory engagement and interactivity. By leveraging visual, auditory and haptic stimuli, VR aims to stimulate all human senses, engaging the user in a digital environment as if it were the real world. Using hand-held controllers, eye-trackers etc., the user is able to interact with the virtual world actively, and in real-time. This enhances realism and helps guide the user's attention within the digital space.

Immersion directly influences gaze behaviour, as users naturally focus their attention on key elements within the virtual environment. The more immersive the experience, the more instinctively users engage with their surroundings, allowing gaze tracking to reveal how attention is distributed and how users interact with the virtual space in real-time.

2.1.4 Foveated Rendering

Foveated (or gaze-contingent) Rendering is an advanced graphics technique designed to optimize rendering performance in virtual reality environments by taking advantage of the human visual system's characteristics. The human eye, as we will describe in detail in Section 2.2, has a high-resolution foveal region at the center of the visual field, where detail perception is sharpest, and a peripheral region where sensitivity to motion and light is greater, but detail perception is lower. Foveated Rendering capitalizes on this by rendering high-quality graphics primarily in the user's gaze area while reducing the detail in peripheral vision areas, where less visual acuity is needed [29].

This technique can significantly improve performance by reducing the workload on the GPU, allowing for higher frame rates and more complex scenes without compromising the user's experience. Foveated Rendering is typically achieved using eye-tracking technology, which detects the user's gaze direction and adjusts rendering accordingly [30].

Foveated rendering is a promising technology, but it still faces several technical challenges and limitations. Foveated rendering is still limited by hardware constraints. Any latency between eye-tracking input and rendering adjustments can lead to "perceived lag," where the high-quality region doesn't align with the user's instantaneous gaze point, reducing the sense of immersion and potentially causing discomfort. Future improvements in eye-tracking technology, rendering hardware, and perceptual modeling are anticipated

to address these challenges further, allowing for more seamless integration of foveated rendering in VR and AR applications [29].

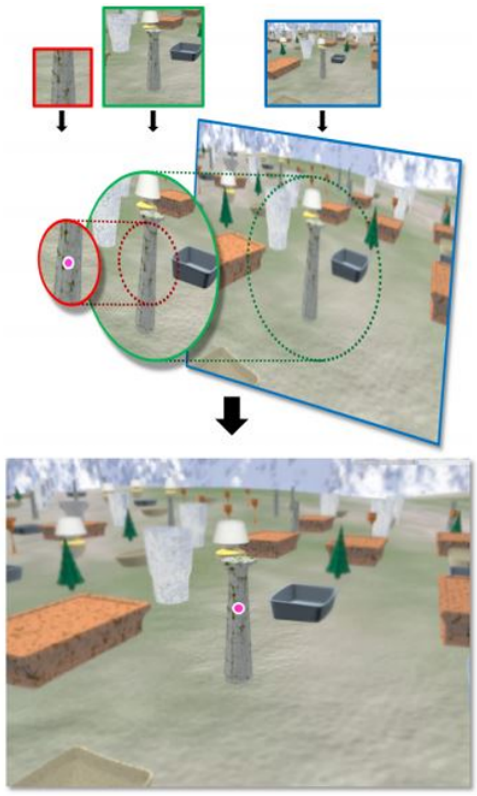


Figure 2.9: Foveated Rendering Example [8]

In this thesis, we aim to develop a gaze prediction model that operates swiftly enough to potentially serve as an alternative to physical eye trackers in HMDs. This would reduce the need for dedicated eye-tracking hardware in each HMD, eliminating the latency issues that sometimes affect these devices. However, a more realistic outcome is that the model could work alongside the eye tracker, predicting the next likely region of user gaze. This collaborative approach would allow the model to assist in directing computations more efficiently, supporting eye-tracking performance with anticipatory gaze localization.

2.2 Human Eye

In this section, we will discuss the fundamentals of the human eye and its role in gaze behaviour. Understanding the anatomy and functionality of the eye is crucial for creating accurate and effective gaze prediction models. The basic anatomy of the eye will be explored, the way it perceives the world, and the different types of eye movements that contribute to gaze behaviour. Additionally, we will examine how gaze behaviour manifests in real-world settings, desktop environments, and virtual reality. This foundational knowledge will provide the context necessary for understanding the challenges and opportunities in developing gaze prediction models for VR environments.

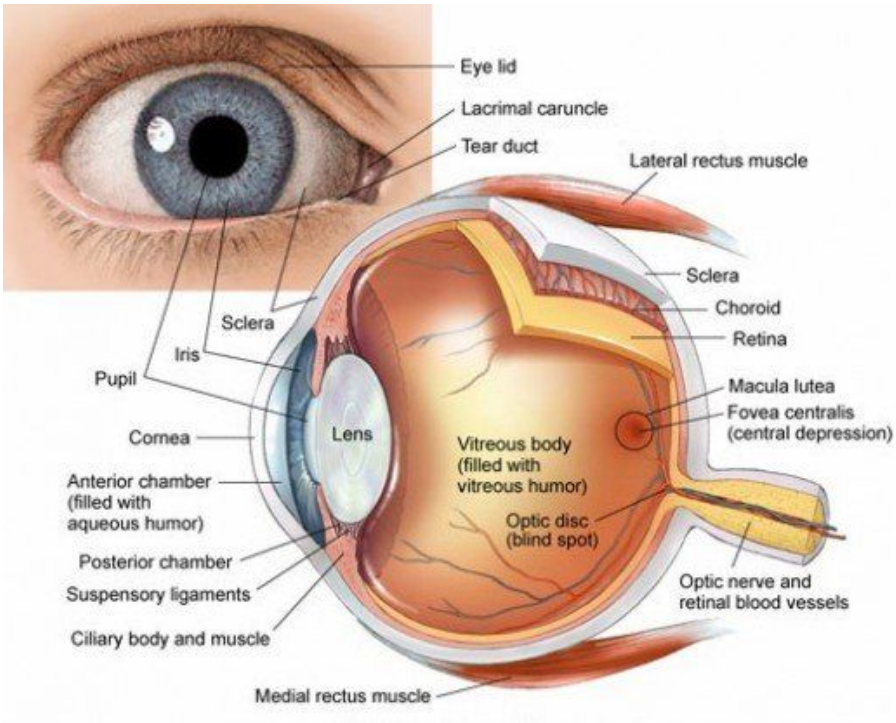


Figure 2.10: Eye Structure [9]

2.2.1 Anatomy of the Eye

The human eye is a complex organ that enables vision by detecting and processing light. It consists of several key structures.

The **cornea** is the transparent, outer "window" and primary focusing element of the eye. The outer layer of the cornea is known as epithelium. Its main job is to protect the eye. The epithelium is made up of transparent cells that have the ability to regenerate quickly. The inner layers of the cornea are also made up of transparent tissue, which allows light to pass.

The **pupil** is the dark opening in the center of the colored iris that controls how much light enters the eye. The iris functions like the iris of a camera, opening and closing to control the amount of light entering through the pupil.

The **lens**, located behind the pupil, further focuses light onto the retina by changing shape (a process known as accommodation).

The **retina** is the membrane lining the back of the eye that contains photoreceptor cells. These photoreceptor nerve cells react to the presence and intensity of light by sending an impulse to the brain via the optic nerve. The multitude of nerve impulses received from the photoreceptor cells in the retina are assimilated into an image in the brain. The **fovea** is the most central part of the retina. This area is responsible for the clearest vision with sharpest colors and details [31, 32].

2.2.2 Vision

Light enters the eye through the cornea and pupil, is focused by the lens, and then strikes the retina. The photoreceptor cells convert light into electrical signals, which are sent to the brain via the optic nerve. The brain processes these signals to create visual images, enabling us to perceive our surroundings.

Vision is not uniform across the visual field; it is sharpest in the center (foveal vision) and decreases in clarity towards the periphery (peripheral vision). This variation in visual acuity influences where and how we focus our gaze [33].

2.2.2.1 Types of Vision

The human visual system involves both monocular and binocular vision. Monocular vision refers to the visual perception of each eye independently, providing a wider field of view but less depth perception. Monocular cues, such as relative size, texture gradient, motion parallax, and linear perspective, assist in judging distances and understanding spatial relationships in the environment [34]. Binocular vision, in contrast, occurs when

both eyes work together to focus on a single point in space. This type of vision is essential for depth perception, allowing us to perceive three-dimensional (3D) structures more accurately. The overlapping visual fields of both eyes create a binocular field of view, which is smaller than the total monocular field but offers the advantage of depth perception through stereopsis [35].

Stereopsis is a key aspect of depth perception and results from the slight differences in the images perceived by each eye, known as binocular disparity. The brain fuses these two slightly different images into a single three-dimensional perception. This process enables the accurate judgment of distances and spatial orientation, which is essential for tasks requiring precise depth perception, such as catching a ball or navigating through complex environments [36]. Stereopsis is most effective at short to medium distances, where the disparity between the two images is most pronounced.

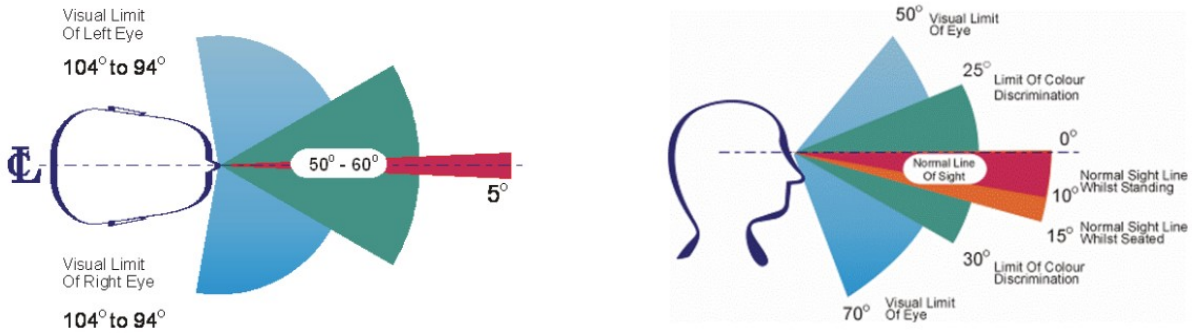
2.2.2.2 Field Of View

The field of view (FOV) of the human eye is quite extensive. Considering both eyes together, the horizontal field of view ranges from approximately 200 to 220 degrees, while the vertical field spans around 135 degrees. The binocular overlap, where both eyes' fields converge, is narrower, covering about 114 to 120 degrees horizontally. This overlap is critical for effective stereopsis and depth perception. The remaining peripheral vision outside this overlap is covered by monocular vision, which plays a crucial role in detecting motion and providing a broad awareness of our environment [37]. Peripheral vision is highly sensitive to movement and is a critical component of our visual attention system, helping with tasks like navigation and hazard detection [38].

By understanding these foundational aspects of vision—monocular and binocular cues, stereopsis, and the field of view—we can better appreciate the complexity of gaze behaviour and the factors influencing where and how individuals focus their gaze in different environments.

2.2.3 Eye Movement and Control

Eye movements are essential for gathering visual information from the environment. The human visual system employs several types of eye movements, including fixations, saccades, smooth pursuit, vergence, and vestibulo-ocular movements, to achieve a stable



(a) Horizontal FOV (b) Vertical FOV

Figure 2.11: Field of View [10]

and clear perception of the world. Understanding these movements is vital for interpreting gaze behaviour and predicting where an individual is likely to look next, which is fundamental for gaze prediction models.

Fixations are the moments when the eyes remain relatively stable and focused on a single point in the visual field. During a fixation, visual information is actively processed, allowing the brain to analyze the details of the object or scene being observed. The duration of fixations can vary significantly, typically ranging from 200 to 600 milliseconds, but they can be shorter or longer depending on various factors such as the complexity of the scene, the cognitive load of the task, and individual differences [39, 40]. Rather than focusing solely on a fixed time frame, fixation is often better defined by the concept of low gaze velocity—when the eye movement speed drops below a certain threshold, it is considered a fixation [41].

The variability in fixation duration and its dependency on visual and cognitive demands make it a key component in gaze prediction models. Areas with longer fixations often indicate higher interest or cognitive processing requirements. Therefore, understanding fixation patterns, including their variability, is essential for accurately predicting areas of interest [42].

Saccades are rapid, ballistic eye movements that occur between fixations, enabling the eyes to shift their focus from one point to another within the visual field. Saccades

serve to bring new areas of interest into the foveal region for detailed processing [43]. These movements are characterized by high gaze velocity, which can reach up to 900 degrees per second, and their duration can range from 20 to 200 milliseconds [39]. Saccades vary widely in duration and velocity depending on the distance between fixation points, the complexity of the visual scene, and the visual search task being performed.

Visual information is suppressed during saccades, a phenomenon known as saccadic suppression, which helps to prevent the blurring of visual information caused by the rapid movement of the eyes. Saccadic movements are essential for visual tasks such as reading, where they help shift focus from one word or line to the next, and for scanning environments where they quickly direct attention to areas of potential interest [42].

For gaze prediction, understanding the dynamics of saccades, including their initiation, velocity, and endpoints, is crucial, especially in dynamic environments such as Virtual Reality, where visual stimuli are constantly changing. The concept of gaze velocity is crucial in distinguishing saccades from fixations [44], as saccades involve high-velocity movements, whereas fixations involve low-velocity or near-static positions of the eyes [40].

In addition to fixations and saccades, other types of eye movements like **smooth pursuit** (used to follow moving objects), **vergence** (adjustments for depth perception), and **vestibo-ocular movements** (stabilizing the gaze during head movements) also play essential roles in vision. However, for gaze prediction, understanding fixations and saccades is particularly important due to their dominant role in scanning and exploring dynamic visual scenes.

2.3 Gaze Behaviour Analysis

Understanding gaze behaviour is critical for developing accurate models of visual attention, especially in dynamic environments like VR. This section explores several fundamental concepts, including saliency, bottom-up and top-down mechanisms, the preferred field of view (FOV), horizontal and vertical independence, different biases, the influence of a task versus free viewing conditions, and head-to-gaze correlation in VR.

2.3.1 Attention Mechanisms

Visual attention is usually modelled with two mechanisms: bottom-up and top-down attention. The bottom-up attention is based on salient features of the input image. Salient features attract attention due to their properties, such as color, brightness, contrast, orientation, and motion [45]. Without the cues or guidance of prior knowledge and in the absence of a specific task, salient regions, meaning locations with high intensity contrast, fresh colour, object edged and motion, predominantly dictate gaze patterns [46]. For example, flashing points of light on a dark night, sudden motion of objects in a static environment, a bright red object in a predominantly green field will naturally draw attention [47]. This mechanism is driven by sensory input and it typically involuntary and very fast. For this reason, it is often called stimuli-driven attention.

Top-down attention refers to the set of processes used to bias visual perception based on task or intention. This mechanism is driven by the mental state of the observer or cues they have received [47]. When users engage in a specific task, their gaze behaviour is guided more by intention and less by the sensory attributes of the stimuli. Top-down processes are particularly important in VR, where immersive environments require focused and task-oriented gaze patterns [18].

2.3.2 Gaze Patterns and Spatial Biases

Research shows that in many settings, gaze is not randomly distributed but instead exhibits predictable patterns due to several visual and cognitive factors.

Firstly, studies prove that viewers tend to fixate on the center of their visual field, both in the natural world, as well as in desktop or VR environments [48, 49]. This phenomenon is called the **center bias**, and is thought to arise from several factors.

One of the primary explanations comes for the fact that image features tend to be biased toward the center of natural images and fixations are correlated with image features [48]. Another explanation is related to the tendency of photographers or content creators to position objects of interest at the center of the image. Also, when people repeatedly watch images with salient information placed in the center, they naturally expect to find the most informative content of the image around its center [49]. Another important reason that encourages this behaviour is the interestingness of the scene.

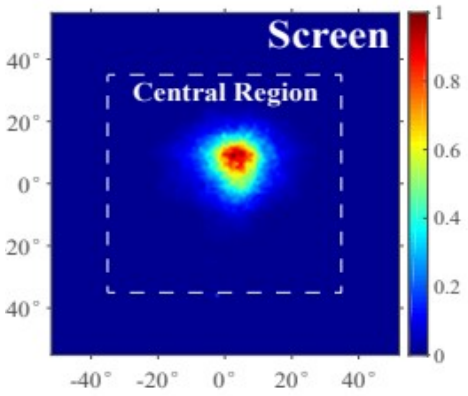


Figure 2.12: DGaze: The distribution of users' gaze position on an HMD screen. 98.7% of the gaze data lies in the central region of the screen. The central region is a square region that is confined to [-35°,35°]×[-35°,35°] in the domain of gaze position [11].

When there are no highly salient regions, humans are inclined to look at the center of the image [17].

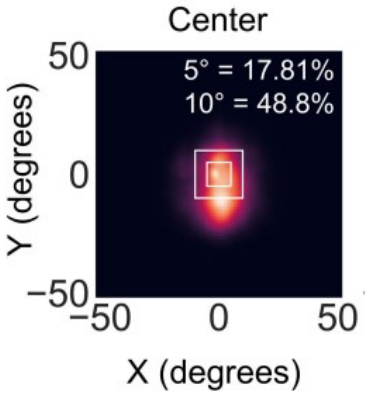


Figure 2.13: Another example of the center bias in the OpenNEEDS dataset [12] .

In VR environments, this center bias becomes more pronounced due to the nature of head-mounted displays (HMDs). Users in VR are more inclined to move their heads to bring information to the center rather than using eye movements alone. This behaviour is partly due to the immersive experience provided by HMDs, where head movements are more natural and less fatiguing than extensive eye movements [18]. The same study presented the **equator bias**, which refers to the tendency of users to focus their gaze along the horizontal midline of a visual screen. This is particularly evident in VR environments

where the equator (horizontal plane) aligns with users' natural head movements, resulting in more frequent gaze activity along this axis.

Another observation addressed often is the **independence of horizontal and vertical eye movements** [11]. Horizontal movements, being more common and less constrained, often dominate gaze behaviour. On the other hand, vertical movements can be more deliberate, as they are less frequent and often task-dependent [18].

Head-to-gaze correlation is an equally important concept in the study of visual attention, especially in VR environments. There exists a range within which gaze positions have a strong linear correlation with head rotation angular velocities [50]. While the eyes can move independently of the head to a certain extent (known as the oculomotor range), when a target is more than 15-20 degrees away from the central line of sight, people tend to move their heads to bring the target closer to the center of their visual field, rather than rely solely on eye movements [51]. This coordination is crucial for maintaining comfort and reducing strain, as large eye movements can be tiring and less efficient.

Finally, gaze behaviour is significantly influenced by the presence of a specific task and the nature of it. Apart from the already mentioned fact that when users engage in a specific task, their gaze behaviour is guided more by intention and less by the sensory attributes of the stimuli, it is important to discuss how the nature of the task itself influences gaze. For example, in first-person shooter games, gaze tends to be more focused on the center of the screen than in adventure games [52]. Even while viewing the same image, the gaze distribution is different according to the task performed by the user, as seen in Figure 2.14.

2.3.3 Temporal Characteristics of Visual Attention

Reaction time is a critical component of visual attention, referring to the delay between the presentation of a stimulus and the user's response to it. Users often exhibit a lag behind moving objects or stimuli due to the time it takes for the brain to process visual information and initiate motor responses [53]. Research has proven experimentally that not only realtime object positions but also past object positions are correlated with gaze positions and thus they can be both applied to the task of gaze prediction [11].

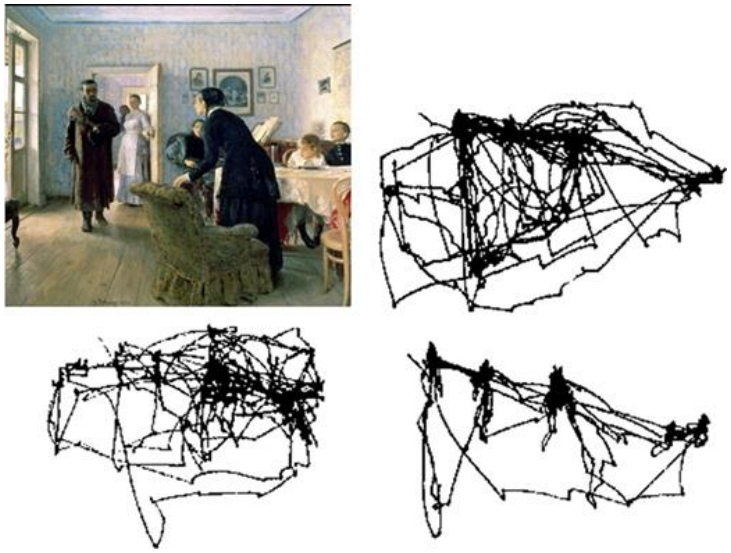


Figure 2.14: Eye trajectories measured by Yarbus (1967) by viewers carrying out different tasks. (Upper right) No specific task. (Lower left) Estimate the wealth of the family. (Lower right) Give the ages of the people in the painting [13].

Finally, another important concept in the field of gaze behaviour is the temporal continuity of visual attention. It is defined as the continuity and consistency of users' on-screen gaze position sequences [54].

In this thesis, the temporal continuity of visual attention is explored deeply. Our model is based on this concept. Our purpose is to examine whether or not temporal continuity is enough to make accurate gaze predictions.

2.3.4 Eye Tracking

We explored the anatomy of the eye and the basics of gaze behaviour. Much of what we understand today about these natural processes comes from the use of eye tracking technology, which has been instrumental in measuring and analyzing where people direct their attention.

Eye tracking is an experimental method of recording eye motion and gaze location across time and task. Most modern eye trackers are video-based. They shine some light source into the eye, usually an infrared light that is invisible to humans. This light produces a reflection on the cornea that is identified by the eye tracking software. The

center of the pupil is also identified by the software. Then a calibration is performed, where the participant is instructed to look at a series of points at known locations on the screen. This calibration is tested in a validation stage. If the calibration is good, the point of gaze (where the participant is looking) can then be estimated with a high degree of accuracy from the relative positions of the pupil and corneal reflection [55].

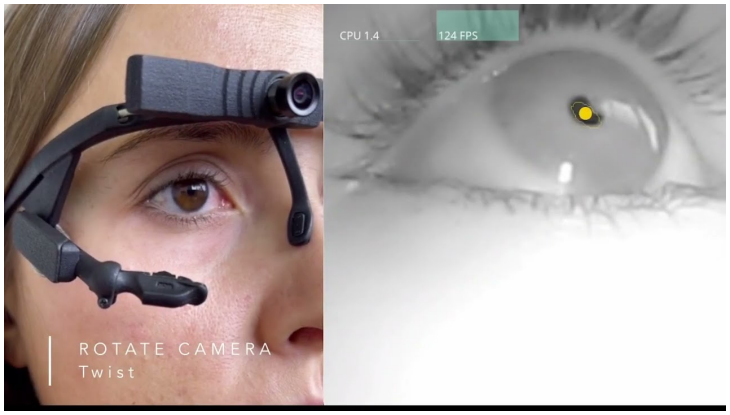


Figure 2.15: Pupil Labs Core Eye Tracker [14].

2.3.4.1 Types of Eye Tracking Systems

There are currently several types of eye trackers. First, there are remote eye trackers, such as the Tobii Pro Fusion [56], which are positioned at a distance from the user, typically on a monitor or screen, and detect gaze without physical attachment. Next, there are wearable eye trackers, such as the Pupil Labs Core [14], shown in Figure 2.15, which are usually glasses or headsets equipped with cameras to track eye movement as the user interacts with the environment. Finally, there are embedded eye trackers, integrated into devices like smartphones, tablets and VR/AR headsets. For example, the HTC Vive Pro Eye headset, shown in Figure 2.8, has an embedded Tobii eye-tracker.

2.3.4.2 Challenges

Although improvements in technology have made eye tracking more affordable and accessible for both users and researchers, there are still challenges that need to be addressed.

Calibration Sensitivity: Eye trackers often require precise calibration for each individual user. Calibration processes can be time-consuming and sensitive to factors such as head position, glasses or contact lenses, and even lighting conditions.

Limited Accessibility: Some individuals with eye-related conditions (e.g., drooping eyelids, nystagmus) may find traditional eye trackers less effective or even unusable. Inconsistent performance across a diverse population limits the broader applicability of these systems.

Hardware Costs: Even though modern eye trackers are becoming more and more affordable, they are still expensive. The hardware involved—cameras, IR illumination systems, and processing units—adds significant cost to devices which are already expensive.

Latency issues: Although modern eye trackers are fast, they still introduce some latency, which can be problematic in real-time applications like virtual reality or gaming. Even a small delay between the user's gaze and system response can disrupt immersion and cause discomfort or motion sickness.

Privacy concerns: Eye trackers collect sensitive data regarding where and how long a user looks at certain objects or areas. This information can inadvertently expose private or unconscious thoughts, raising privacy concerns when used in commercial or surveillance applications.

Given these limitations, recent advances in machine learning and neural networks offer a promising alternative: gaze prediction models.

2.4 Gaze Prediction

In this section, we will explore the research surrounding gaze prediction, examining its origins and the foundational works in the field. We will analyze key architectural structures that have proven essential for the task of gaze prediction and examine the state-of-the-art models for gaze prediction in virtual reality. Based on this research, several important factors emerged that must be considered when developing a gaze prediction model, along with areas that require further attention, which will guide the focus of our implementation.

2.4.1 Early Steps

Roughly two decades ago, Laurent Itti and Cristoph Koch presented their model of bottom-up visual attention [45] based on the cognitive theoretical foundations of human visual attention. The model subsequently became a seminal work in the field, inspiring a multitude of researchers from various domains to propose their own models of visual attention. The Itti and Koch model was so influential because it combined several different aspects to reach one goal: to understand human visual processing by simulating the processing stages from scene level input to fixation selection. The model had a strong basis in cognitive theory and tried to replicate some neuronal mechanisms involved in visual attention by using many of the best approaches from computational vision available at the time. The result was a model that was to account for many of the spatial and the temporal aspects of human shifts of visual attention [57].

In this paper, they model gaze prediction using a saliency map, whch is defined as a topographic representation that encodes the conspicuity of visual stimuli across a scene, guiding attention to the most salient locations based on low-level visual features such as color, intensity, and orientation. This representation of visual attention became a standard approach, with subsequent models attempting to predict saliency maps.

2.4.2 Predicting Saliency Maps

With the evolution of deep learning, newer models have emerged. Using deep neural networks, more complex information can be exploited, such as high-level features, leading to more and more top-down approaches for the task of predicting gaze and fixations.

2.4.2.1 The VGG19 Network

The **VGG19** network [58], introduced by Simonyan and Zisserman in 2014, is a deep convolutional neural network (CNN) that has become a standard architecture for image classification and feature extraction tasks. It consists of 19 layers, including 16 convolutional layers followed by fully connected layers (further analysis in section 3.1.4). The primary strength of VGG19 lies in its simplicity: it uses small 3×3 convolution filters, allowing the network to stack many layers while maintaining manageable computational complexity. This depth helps the network learn rich hierarchical representations, from low-level features like edges and textures to high-level semantic concepts.

In the context of gaze prediction, VGG19 is often used to extract high-level features from images, which can then be fed into saliency prediction models.

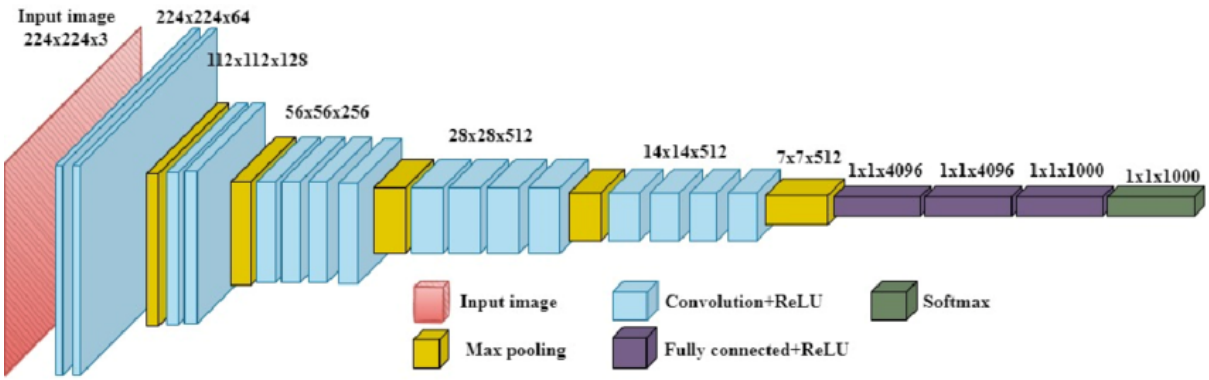


Figure 2.16: Representation of the VGG-19 architecture [15].

A known model for saliency prediction based on the VGG19 network is DeepGaze II [59]. The key innovation of DeepGaze II is its use of deep features extracted from the VGG19 network, which was originally trained for object recognition tasks. By utilizing a pre-trained VGG19 model, DeepGaze II captures rich visual features from input images that are crucial for predicting human fixation points, i.e., where people are likely to focus their attention. This model is particularly robust because it generalizes well across various image datasets without needing extensive retraining or fine-tuning.

Another state-of-the-art model using the VGG19 network is SALICON [16]. Salicon incorporates both global and local context by using multi-resolution image input and feature extraction. The extracted features from different scales are then combined to create a saliency map, predicting the likelihood of each pixel in the image being a fixation point. The final prediction is refined using techniques like deconvolution or upsampling to ensure that the output saliency map has the same spatial dimensions as the input image.

Salicon's innovation lies in its ability to predict saliency at a large scale efficiently, using pre-trained deep networks like VGG for feature extraction and combining multi-scale image analysis. This allows it to capture both local attention cues and broader scene context, leading to accurate saliency predictions that closely match human eye movements.

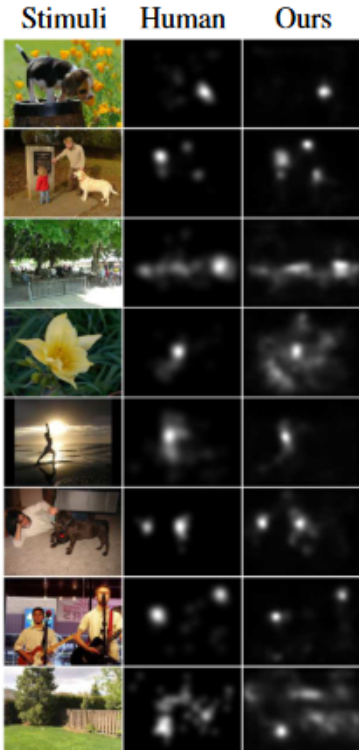


Figure 2.17: SALICON model results [16]. The model can effectively detect salient regions with different semantic content, and different sizes.

2.4.2.2 ResNet50

ResNet50 [60], introduced by He et al. in 2015, represents a major advancement in CNN architecture through the concept of residual learning. Traditional deep networks, like VGG19, tend to suffer from degradation problems as layers are added, leading to difficulties in optimization. ResNet solves this by introducing "skip connections," which allow the network to bypass certain layers, effectively creating shortcuts. This enables ResNet50, which has 50 layers, to train very deep networks without suffering from the vanishing gradient problem. This structure has been used in numerous models for gaze prediction. For example, one well-known is the **SAM-ResNet50** model [17].

This model became well-known in the field of visual attention and gaze prediction due to its innovative combination of a ResNet50 backbone for spatial feature extraction and an LSTM-based attention mechanism for temporal and iterative saliency prediction. It outperformed many previous methods in predicting human eye fixations by focusing on

both the spatial saliency and temporal dynamics of how attention shifts across images. This integration of spatial and temporal modeling, along with the attention mechanism, allows the model to predict human eye fixations more accurately than traditional feed-forward approaches. The model iteratively updates the saliency map, simulating how humans shift their gaze to different parts of an image.

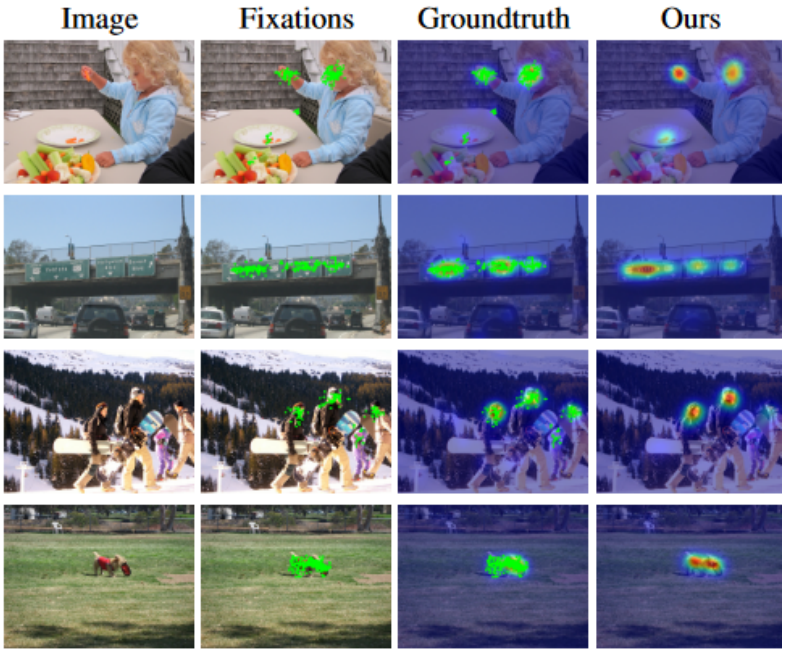


Figure 2.18: SAM-ResNet results [17].

2.4.3 Gaze Prediction in Virtual Reality

Gaze prediction in virtual reality is gradually gaining more attention. Researchers are increasingly studying the differences between natural environments, desktop viewing conditions, and VR. Several factors must be taken into account, such as varying perspectives, the relationship between head movement and gaze, more dynamic content, and the increased complexity of tasks involved.

Sitzmann et. al [18] analyzed the way people explore immersive virtual environments and tested several existing saliency predictors to immersive VR conditions. Their work highlights several key facts. Using the Pearson Correlation (CC) score, which ranges from -1 (perfectly inversely correlated) to 1 (perfectly correlated), they compared VR to

desktop conditions and got a CC score of 0.76, which means that although different, there is common ground, which means that they could try and use existing saliency predictors for desktop images for VR gaze prediction. They also mention the existence of the equator bias. Thus, their model is based on existing and successful saliency predictors [61, 62], which they use to get a saliency map. They then multiply this map with the longitudinal equator bias and get a final prediction.

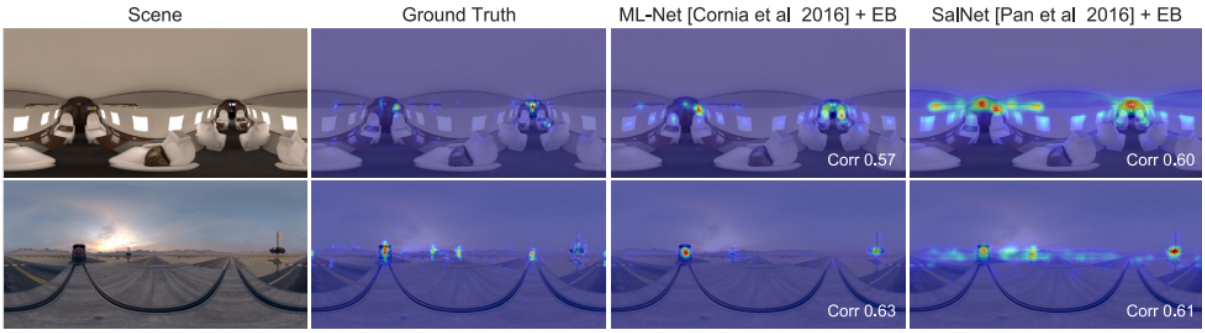


Figure 2.19: Saliency prediction for omni-directional stereo panoramas [18]. Using existing saliency predictors along with the equator bias, good results are achieved.

Their results were promising, but worse for VR than for desktop environments. Also, it is worth mentioning that the experiment was conducted in a static VR environment, with neither dynamic objects nor dynamic usage by the user.

Koulieris et al. [63] focused on a heavily task-oriented game and tried to predict gaze. Their observation is that player actions are highly correlated with the present state of a game, encoded by game variables. Based on this, they trained a classifier to learn these correlations using an eye-tracker which provided the ground-truth object being looked at. The classifier can be used at runtime to predict object category – and thus gaze – during game play, based on the current state of game variables. However, they focused on First Person Shooter (FPS) games.

A big breakthrough in the field is the model **DGaze** [11], by Hu et al. First of all, they highlight the fact that gaze prediction is different from prior works in human eye fixations or visual saliency, which predict a density map of eye fixations. In contrast, real-time gaze prediction aims to predict a single gaze and the real-time requirements (60Hz or better) make it distinct from approaches that focus on saliency prediction. Especially for some VR applications like gaze-contingent rendering and eye movement interactions,

a user's real-time gaze position has more practical significance than a density map of eye fixations.

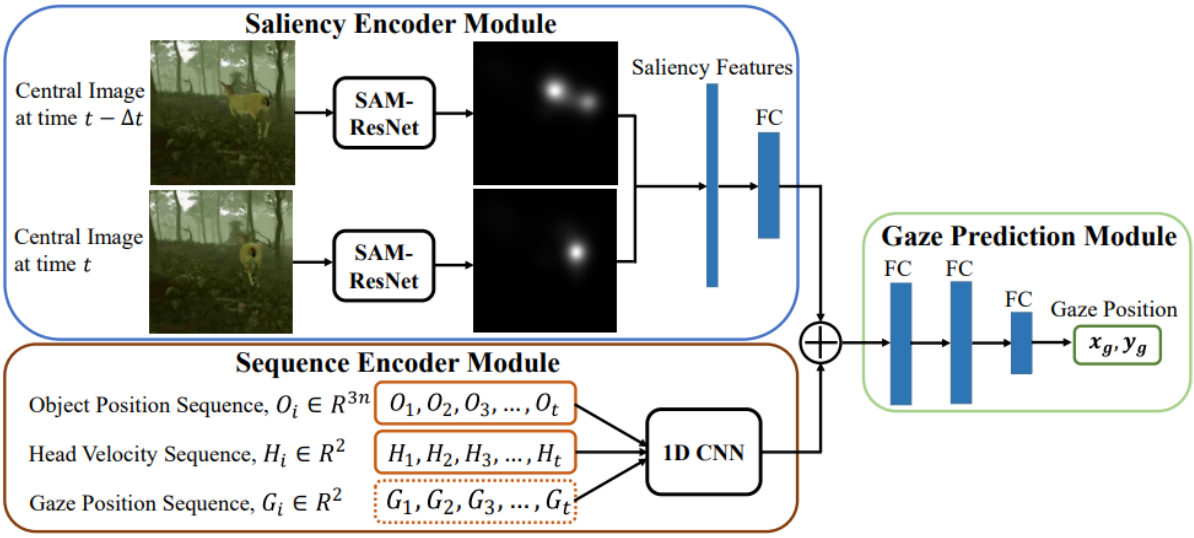


Figure 2.20: DGaze architecture.

Moreover, they use dynamic virtual scenes to collect data, train their model, and also test it. This is very important, as they begin to study more complex VR environments, with dynamic objects. Their model is based on three modules:

Saliency Encoder Module: Utilizing Sam-ResNet [17], they extract the saliency maps of the frames every 250 ms, and using a FC layer they encode the saliency features.

Sequence Encoder Module: They encode the sequence of dynamic object prosi-tions, head velocities and if available, the sequence of past gaze positions, employing a 1D CNN layer, to predict gaze positions.

Gaze Prediction Module: Finally, they incorporate the outputs of the two mod-ules, using 3 FC layers to make a single gaze point (x_g, y_g) prediction.

Finally, this work paves the way to a different evaluation approach. They present some new evaluations metrics, appropriate for real-time gaze prediction: they use the angular distance as an error metric, calculating the angle between the predicted and the groundtruth data. They compare their results with two baselines, deriving from gaze behaviour analysis: the center baseline (i.e. assume the model predicts always the center,

as this is where users are most likely to look at), and the mean baseline (i.e. the mean gaze point, calculated from the gaze data they created).

Their results are quite promising, and this work has influenced the field in multiple ways. However, there are still some limitations. Their model is derived from free-viewing conditions (no specific task) and performs worse on task-oriented situations. Moreover, their system requires a complex input, which means that it is not easily adaptable from application to application.

Hu et al. [54] presented the concept of temporal continuity of visual attention in immersive virtual reality and evaluated it in both free-viewing and task-oriented conditions, by calculating autocorrelation functions of users' gaze position sequences. They further applied the temporal continuity to the task of future gaze prediction, utilizing current gaze positions to predict gaze positions in the future. Their work revealed that, in both free-viewing and task-oriented conditions, temporal continuity can only efficiently facilitate short-term gaze prediction. However, their work was mainly a proof of concept, and no complex model based on temporal continuity was created.

Finally, Hu et al. presented **FixationNet** [64], a novel model for forecasting human eye fixation in task-oriented virtual environments. They focused on the task of visual search, which requires subjects to detect a target among many distractors. Their architecture was similar to DGaze, and the question they mainly tried to answer is the time interval in which their model's prediction is accurate. Once again, it is proved that prediction works for the short-term future. It is important to note that this work proved that gaze prediction on task-related situations is possible. However, their model focused on a very specific task, and cannot be directly applied to other kinds of tasks.

2.4.4 Limitations—Requirements

Our research has revealed several key aspects related to the task of gaze prediction. First, there is a need to focus on prediction for task-specific environments in virtual reality. Second, a more generalized model is necessary—one that can adapt and predict accurately across various task categories.

To ensure compatibility with existing systems, it is important to simplify the input as much as possible, minimizing the need for extensive scene information (such as game

variables, objects, saliency, etc.). This will enhance the model's flexibility, allowing it to be easily integrated into different systems.

Additionally, for this approach to be practical, the system must be lightweight and capable of making predictions rapidly, ensuring it can be used in gaze-based interactions or gaze-contingent rendering pipelines.

Regarding the architecture of the model, we believe that task-specific environments and the concept of temporal continuity are closely related. Further exploration of this relationship is needed to develop a more accurate model.

Chapter 3

Theoretical and Technological Background

In this chapter, we will present the theoretical and technological background essential for our implementation. We will start by analyzing the theory of neural networks, including their components, with a focus on the layers and techniques used in this thesis. Next, we will discuss key concepts such as the 2D visual angle. Finally, we will provide an overview of the file formats and software tools utilized in our work.

3.1 Deep Learning

Deep learning is a subset of machine learning that focuses on using neural networks with multiple layers to model complex patterns in data. These neural networks consist of layers of interconnected "neurons," which process inputs and pass information through non-linear transformations [65]. Unlike traditional machine learning models, deep networks are able to automatically learn feature representations, making them highly effective in tasks like image classification, speech recognition, and gaze prediction [66].

3.1.1 Neural Network Basic Structure

Artificial Neural Networks consist of an input layer, multiple hidden layers, and an output layer [67]. Each layer processes information received from the previous layer and passes it on to the next. Layers consist of interconnected nodes (neurons) that process input data

and produce output predictions. For a neural network to be considered deep, it typically must consist of at least three layers [66].

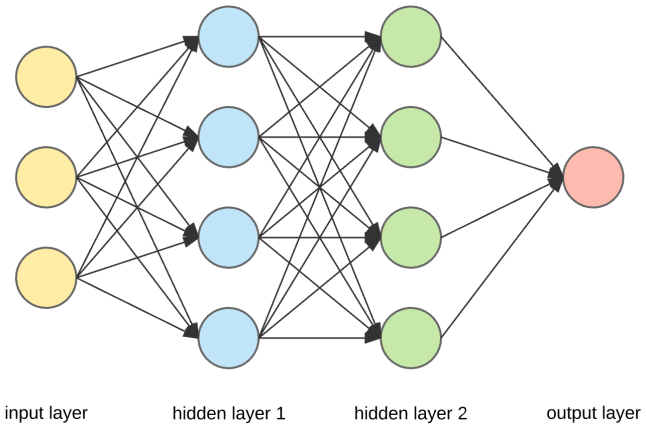


Figure 3.1: ANN example [19]

A **neuron** is the fundamental building block of a neural network. It is inspired by the structure and functioning of biological neurons in the human brain. Artificial neurons are designed to process and transmit information in a neural network, enabling the network to perform complex tasks such as pattern recognition, decision-making, and learning.

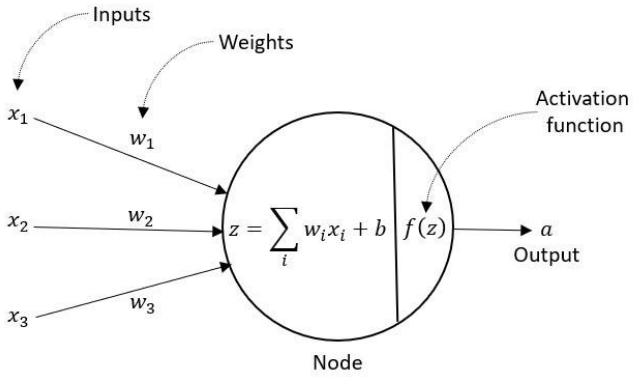
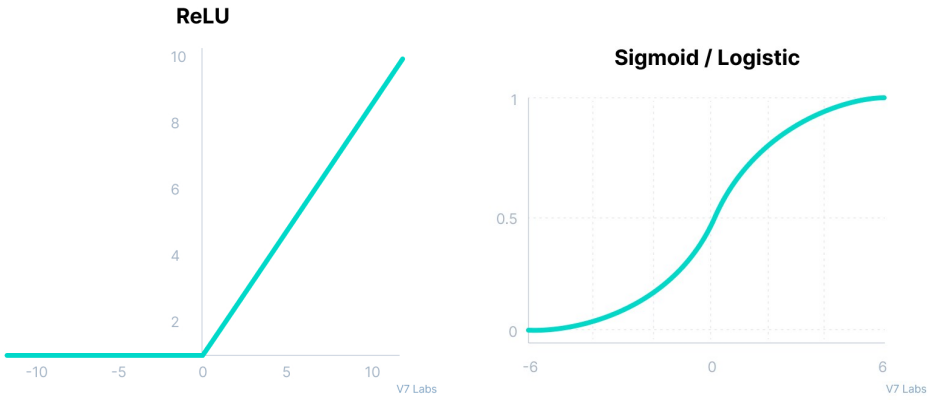


Figure 3.2: Diagram of an artificial neuron [20]

- A neuron receives n input signals represented as $x_1, x_2, ..., x_n$.
- For each input signal, there are weights represented as $w_1, w_2, ..., w_n$.

- A bias term represented as b is added to the summation of weighted inputs.
- The weighted inputs and the bias term are added together to produce a weighted sum: $z = \sum w_i x_i + b = x_1 w_1 + x_2 w_2 + ... + x_n w_n + b$.
- The weighted sum z is then passed through an **activation function** *f(x)*, which introduces non-linearity to the neuron's output. This is essential for the model to learn complex patterns. Common activation functions include the sigmoid, ReLU (Rectified Linear Unit), tanh (hyperbolic tangent), and more.
- The *output* of the neuron is the result of the activation function.

In this thesis, two activation functions were used: ReLU and sigmoid.



(a) ReLU [68] (b) Sigmoid [68]

Figure 3.3: Activation Functions

The ReLU function returns 0 if it receives any negative input, but for any positive value x it returns that value back. So it can be written as $f(x) = max(0, x)$.

The sigmoid function takes any real value as input and outputs values in the range of 0 to 1. Mathematically it can be represented as $f(x) = \frac{1}{1 + e^{-}x}$.

3.1.2 Loss Functions

The performance of a network is computed by using a loss or a cost function through, which evaluates how well the network performs in its predictions [66]. The loss measures

the error between the network's output with the actual target in the training data. After the error is calculated, **backpropagation** occurs, where the error is propagated backward through the network to adjust the weights [69]. This process, using techniques like gradient descent, is key to allowing the model to learn.

There is a plethora of loss functions used, depending on the specifics of the model and its goal.

3.1.2.1 Mean Squared Error (MSE)

The Mean Squared Error (MSE) is one of the most commonly used loss functions in regression tasks. It calculates the average of the squared differences between the predicted values and the actual target values [67]. The squaring ensures that large errors are penalized more heavily than small errors.

The formula for MSE is:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \widehat{y}_i)^2$$

Where:

- n is the number of data points.
- y_i represents the true value.
- $\widehat{y}_i$ represents the predicted value.

MSE is sensitive to outliers because the errors are squared, making it useful when we want to penalize large errors more than smaller ones.

3.1.2.2 Mean Absolute Error (MAE)

The Mean Absolute Error (MAE) calculates the average of the absolute differences between predicted values and actual values. Unlike MSE, it doesn't square the errors, making it less sensitive to outliers [70].

The formula for MAE is:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \widehat{y}_i|$$

Where:

- n is the number of data points.
- y_i represents the true value.
- $\widehat{y}_i$ represents the predicted value.

MAE gives an equal weight to all errors, making it more robust to outliers compared to MSE. In this thesis, we experimented with both, but used MAE as the loss function of our final model.

3.1.3 Optimizers

Optimizers adjust the model's weights to minimize the loss function during training. They play a key role in speeding up the learning process and ensuring convergence. In this thesis, we used **Adam (Adaptive Moment Estimation)** [71]. Adam is one of the most popular optimization algorithms due to its efficiency and ability to adapt learning rates.

The **learning rate** is a critical hyperparameter in the training of neural networks, controlling how much the model adjusts its weights in response to the error at each step of optimization. It determines the step size the optimizer takes while traversing the error surface during gradient descent. A small learning rate means that the model will make minor adjustments to the weights, leading to slow but stable convergence. However, if the learning rate is too small, training can become inefficient and might get stuck in local minima. On the other hand, a large learning rate allows the model to update its weights more significantly, which speeds up training but risks overshooting the optimal solution, causing instability or divergence.

3.1.4 Convolutional Neural Networks (CNN)

A Convolutional Neural Network (CNN) is a deep learning algorithm specifically designed for processing and analyzing spatial data, such as images and videos [72].

CNNs are widely used for image classification, object detection, and other vision-related tasks because they excel at recognizing patterns like edges, textures, and shapes,

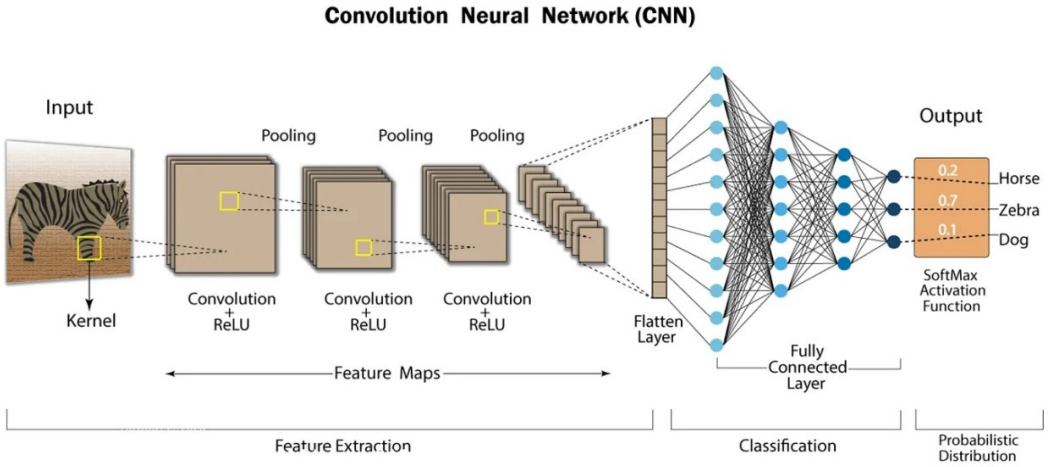


Figure 3.4: CNN typical architecture [21]

which are critical for visual understanding [73]. Unlike traditional fully connected networks, CNNs take advantage of the spatial structure of data, allowing them to handle large input dimensions more efficiently [74]. By using a series of layers designed to progressively extract and refine features, CNNs reduce the need for manual feature engineering. This allows the model to autonomously learn low-level patterns (e.g., edges) in early layers and progressively more complex features (e.g., object shapes) in deeper layers [75].

CNNs consist of several specialized layers, each performing a distinct operation on the input data. The most critical layers in a typical CNN architecture include:

- convolutional layers
- pooling layers
- fully-connected (dense) layers.

3.1.4.1 Convolutional Layer

The convolutional layer is the core building block of CNNs, responsible for extracting features from the input [74]. It applies a set of learnable filters (also called kernels) to the input image or feature map. Each filter slides over the input data, performing a mathematical operation called convolution, and produces an output known as the feature

map. These feature maps capture important local patterns such as edges, corners, or textures.

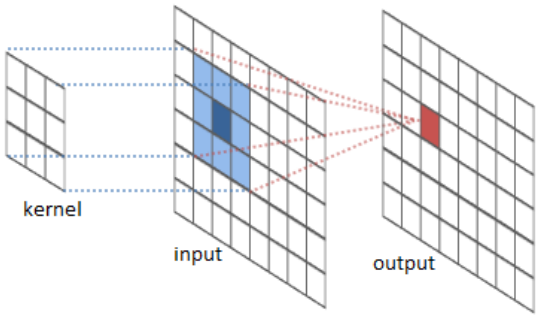


Figure 3.5: Convolution Visualisation [22]

The convolution operation is described mathematically as:

$$FeatureMap(i,j) = \sum_{m,n} Input(i+m, j+n) \cdot Kernel(m,n)$$

Where:

- **Input** is the input image or the previous layer's output (feature map).
- **Kernel** (or filter) is a small matrix of weights that is learned during training.
- **i,j** represent the spatial coordinates of the resulting feature map.

Each filter extracts specific features, and by stacking multiple convolutional layers, CNNs learn increasingly abstract and complex representations of the data [75].

Key Parameters of the Convolutional Layer:

- **Kernel Size:** Determines the size of the filters, typically 3×3 or 5×5 in many architectures [58].
- **Stride:** Defines the step size at which the filter moves across the input image.
- **Padding**: Adds extra pixels around the input, ensuring the output feature map size matches the input dimensions or is reduced in a controlled way.

3.1.4.2 Pooling Layer

After the convolutional layer, the feature map is typically passed through a pooling layer. The pooling layer reduces the spatial dimensions (height and width) of the feature maps, preserving the most important information while making the network computationally efficient and reducing overfitting [76]. The most common form of pooling is **max pooling**, which selects the maximum value from a set of pixels, while average pooling computes the average.

A pooling layer summarises the features present in a region of the feature map generated by a convolution layer. So, further operations are performed on summarised features instead of precisely positioned features generated by the convolution layer. This makes the model more robust to variations in the position of the features in the input images.

3.1.4.3 Fully Connected (Dense) Layer

In the final layers of a CNN, the high-level features extracted by the convolutional and pooling layers are flattened into a 1D vector and passed through one or more fully connected (dense) layers. These layers serve to map the learned features to the final output, such as class probabilities in image classification [73].

3.1.4.4 CNNs in Gaze Prediction

CNNs are widely used in gaze prediction models due to their ability to learn both low-level image features and high-level, complex patterns. This makes them effective for extracting features that influence gaze direction. For instance, as discussed in section 2.4.2.1, models like VGG16 and VGG19 are often components of more complex architectures for saliency and gaze prediction.

In this thesis, we do not directly use a CNN but rather a ConvLSTM network, which builds upon CNN architectures, combining them with LSTM architectures. This will be analyzed in the following sections.

3.1.5 Long Short-Term Memory (LSTM) Networks

LSTMs are a type of recurrent neural network (RNN) specifically designed to model temporal sequences and capture long-term dependencies in sequential data. They were

introduced to address the limitations of traditional RNNs, particularly their difficulty in learning long-term dependencies due to the vanishing gradient problem [77].

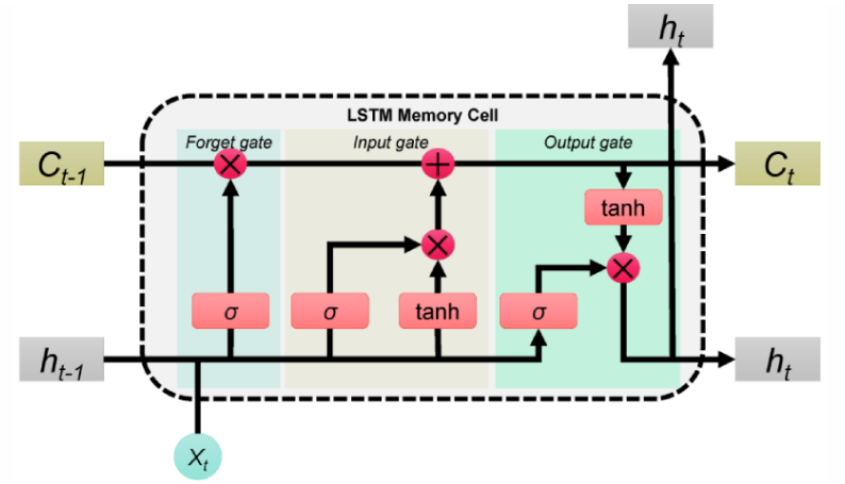


Figure 3.6: LSTM cell structure [23]

LSTMs are composed of memory cells that maintain information over time, and gates that control the flow of information in and out of these cells. This architecture allows LSTMs to selectively retain or forget information over long sequences.

Forget Gate

The information that is no longer useful in the cell state is removed with the forget gate. Two inputs x_t (input at the particular time) and h_{t-1} (previous cell output) are fed to the gate and multiplied with weight matrices followed by the addition of bias. The resultant is passed through an activation function which gives a binary output. If for a particular cell state the output is 0, the piece of information is forgotten and for output 1, the information is retained for future use. The equation for the forget gate is:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

where:

- W_f represents the weight matrix associated with the forget gate.
- $[h_{t-1}, x_t]$ denotes the concatenation of the current input and the previous hidden state.

- b_f is the bias with the forget gate.
- σ is the sigmoid activation function, which outputs values between 0 and 1.

Input Gate

The addition of useful information to the cell state is done by the input gate. It consists of two parts:

- the **input gate**, which decides which values to update.
- the **candidate cell state** ($\tilde{C}_t$), which contains the new potential values to be added to the cell state.

The equation for the input gate is:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

and the cell state:

$$\tilde{C}_t = tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

where:

- W_i, W_C represent the corresponding weight matrices.
- b_i, b_C represent the corresponding bias terms.
- The *tanh* (hyperbolic tangent) function outputs values between -1 and 1, allowing the network to scale new information effectively.

Cell State Update (C_t)

The current cell state C_t is updated by combining the previous cell state C_{t-1} and the new candidate cell state $\tilde{C}_t$. The update is controlled by both the forget gate and the input gate:

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t$$

where:

- the forget gate f_t controls how much of the old cell state C_{t-1} is retained.

- the input gate i_t and candidate cell state $\tilde{C}_t$ determine how much new information to add.

Output Gate (O_t)
The output gate controls what information from the current cell state C_t should be output as the hidden state h_t. This hidden state is used in the next time step and for making predictions. The output gate is calculated as:

$$O_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

where:

- W_o, b_o are the weights and bias for the output gate.

Finally, the hidden state h_t is computed by applying the *tanh* function to the updated cell state C_t, and modulating it with the output gate:

$$h_t = O_t \cdot tanh(C_t)$$

- The hidden state h_t is what is passed to the next LSTM unit in the sequence and is also used as the output of the current time step.

3.1.5.1 LSTMs in Gaze Prediction

In the context of gaze prediction, LSTMs are particularly useful because gaze data is inherently temporal—where a person looks at any given moment is influenced by where they've looked in the past. LSTMs can model these temporal dependencies effectively, capturing both short-term (immediate) and long-term gaze patterns.

In this thesis, we use a LSTM network to predict gaze based on previous sequential gaze points.

3.1.6 Convolutional LSTM (ConvLSTM) Networks

ConvLSTM networks are an extension of LSTM networks designed to model spatiotemporal data. While standard LSTMs handle sequential data effectively, they are not inherently suited to process spatial data (e.g., images or videos). ConvLSTMs, proposed by Shi et al. [24], introduce convolutional operations into the LSTM framework, making

them ideal for handling data that has both spatial and temporal dependencies, such as video frames or sequences of images.

The key innovation in ConvLSTM is that instead of performing matrix multiplications at each gate (as in standard LSTMs), ConvLSTM applies convolutional operations, allowing it to capture spatial features in addition to temporal dependencies. This makes ConvLSTM networks particularly powerful for tasks like video analysis, weather forecasting, and gaze prediction, where understanding both spatial and temporal dynamics is essential.

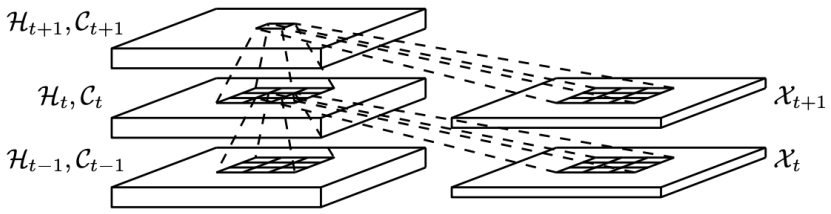


Figure 3.7: Inner structure of ConvLSTM [24]

Similar to the standard LSTM, the ConvLSTM unit consists of gates (forget, input, and output gates), a cell state, and hidden states. However, the ConvLSTM cell operates on 3D tensors instead of 1D vectors, where the third dimension typically represents spatial features (e.g., height, width, and depth of an image). The equations, thus, are almost identical, with the major difference being the convolution instead of matrix multiplication:

$$f_t = \sigma(W_f * [h_{t-1}, x_t] + b_f)$$

$$i_t = \sigma(W_i * [h_{t-1}, x_t] + b_i)$$

$$\tilde{C_t} = tanh(W_C * [h_{t-1}, x_t] + b_C)$$

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C_t}$$

$$O_t = \sigma(W_o * [h_{t-1}, x_t] + b_o)$$

$$h_t = O_t \cdot tanh(C_t)$$

where '$*$' denotes the convolution operation.

The convolutional structure in ConvLSTM ensures that the spatial dependencies are captured at each step, while the LSTM structure preserves temporal information, making ConvLSTM highly effective for modeling spatiotemporal data.

3.1.6.1 ConvLSTMs in Gaze Prediction

In the context of gaze prediction, ConvLSTMs can be particularly useful for modeling how visual attention shifts over time, in correlation to the visual stimuli.

In this thesis, we use a ConvLSTM network to predict gaze points on a current frame based on sequences of past frames. The ConvLSTM network allows us to effectively learn the correlation between sequences of frames and gaze shifts, leading to a more accurate gaze prediction.

3.2 2D Visual Angle

Gaze information provided as a 3D gaze vector (*gazeX, gazeY, gazeZ*) can be transformed to a 2D visual angle (θ, ϕ), using the following formula:

$$\theta = arctan\left(\frac{gazeX}{gazeZ}\right), \phi = arctan\left(\frac{gazeY}{gazeZ}\right)$$

where:

- θ is the horizontal angle between the gaze direction and the forward direction (on the xz-plane).
- ϕ is the vertical angle between the gaze direction and the forward direction (on the yz-plane).

3.3 Min-Max Normalization

Min-max normalization is one of the most common ways to normalize data. For every feature, the minimum value of that feature gets transformed into a 0, the maximum value gets transformed into a 1, and every other value gets transformed into a decimal between 0 and 1, with the following formula:

$$normalized = \frac{value - min}{max - min}$$

3.4 File Formats

In this section, we will mention some file formats that were used in the thesis implementation.

3.4.1 Apache Parquet

Apache Parquet is an open source, column-oriented data file format designed for efficient data storage and retrieval. It provides high performance compression and encoding schemes to handle complex data in bulk and is supported in many programming language and analytics tools [78].

3.4.2 Hierarchical Data Format (HDF)

Hierarchical Data Format version 5 (HDF5) is a data model, library, and file format for storing and managing large and complex datasets [79]. It is designed to address the challenges of handling massive volumes of data efficiently and effectively. HDF5 organizes data into a hierarchical structure, allowing users to organize, manage, and access data in a flexible and scalable manner.

At its core, HDF5 operates as a container format, capable of storing diverse data types, including numeric data, images, text, and metadata. It supports multidimensional arrays, allowing for the representation of complex data structures commonly encountered in scientific and engineering applications. HDF5 files can store vast amounts of data in a single file, making it well-suited for applications that require efficient data storage and retrieval.

3.5 Software Tools

Finally, in thic section, we will present the software tools used in this thesis.

3.5.1 Google Colab

Google Colab, short for Colaboratory, is a cloud-based platform developed by Google that allows users to write and execute Python code in a Jupyter notebook environment.

It provides free access to computing resources, including GPUs and TPUs, making it an excellent tool for machine learning, data analysis, and scientific computing. Colab integrates seamlessly with Google Drive, allowing for easy storage and retrieval of files. With built-in support for popular libraries like TensorFlow, PyTorch, and NumPy, Google Colab is a powerful resource for both beginners and experienced developers looking to experiment with code and data [80].

3.5.2 Tensorflow-Keras

TensorFlow Keras is an open-source machine learning library that simplifies the process of building and training deep learning models. It is part of the TensorFlow ecosystem, offering a high-level API that enables users to design complex neural networks with ease. Keras provides a user-friendly interface for defining layers, compiling models, and training them on large datasets, making it accessible to both beginners and experienced practitioners. With support for various neural network architectures, including convolutional and recurrent networks, TensorFlow Keras allows for rapid prototyping and experimentation [81].

3.5.3 OpenCV

OpenCV (Open Source Computer Vision Library) is a popular open-source library designed for real-time computer vision and image processing tasks. It provides a wide range of tools and algorithms for handling images and videos, such as reading, writing, and manipulating image data. In our project, we use OpenCV to efficiently load and normalize image frames, preparing them for use in deep learning models. Its extensive support for image formats and simple API make it a valuable tool for image preprocessing in machine learning pipelines [82].

Chapter 4

Implementation

Chapter 4 details the implementation of the gaze prediction model. It covers all the essential steps taken to develop the model, including selecting the appropriate dataset, preprocessing the data, designing the system's architecture, and training the model. This chapter provides a thorough overview of the process involved in building our gaze prediction system.

4.1 Dataset

Choosing the right dataset is the first and perhaps most important step in building a robust prediction system. The purpose of this thesis is to create a real-time gaze prediction model for VR task-oriented environments, based on the concept of temporal continuity. This means that that the dataset must fulfill certain requirements:

- First and foremost, the dataset must be **large-scale**. Generally, for a model to make accurate predictions, a large amount of data is required for it to extract the right features and learn patterns. The more challenging the problem, the more training data is needed. Small datasets can be misleading or non-representative.
- Ideally, the dataset should include data from multiple **different users** to avoid user bias, as well as from **various VR scenes** to ensure generalizability.
- For our implementation, where the goal is for the model to predict gaze for a variety of tasks, the dataset must include data from users performing **different tasks**.

- To support the concept of temporal continuity, the dataset must include **continuous data**.

For all these reasons, the dataset chosen for this thesis is **OpenNEEDS** [12], published in 2021 at ETRA '21: 2021 Symposium on Eye Tracking Research and Applications [83]. OpenNEEDS is a large-scale, high frame rate, comprehensive, and open-source dataset of Non-Eye (head, hand, and scene) and Eye (3D gaze vectors) data captured for 44 participants as they freely explored two virtual environments with many potential tasks.

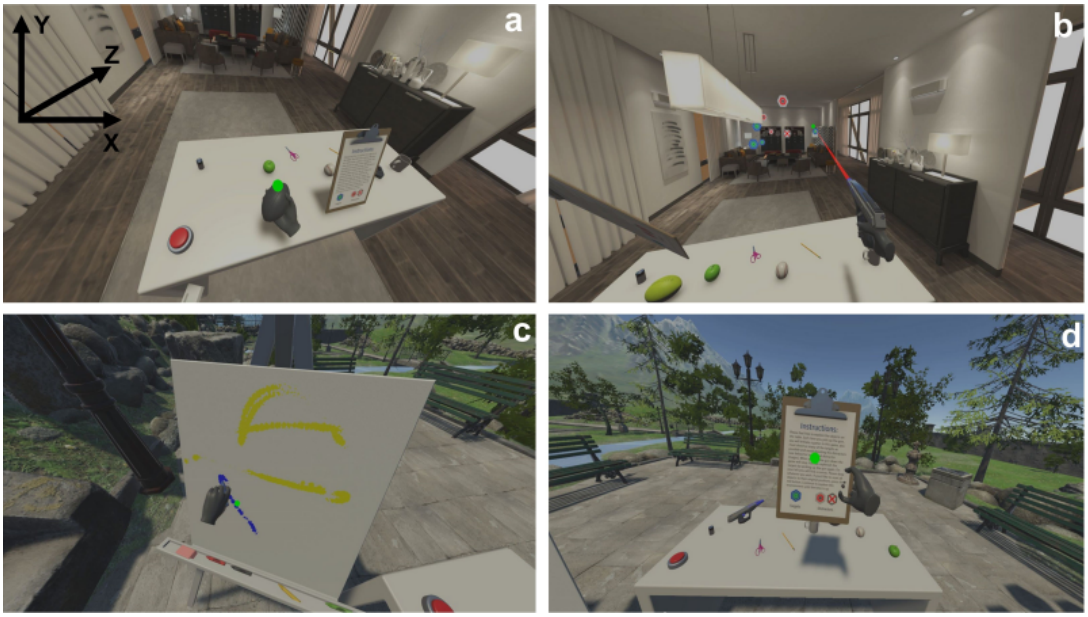


Figure 4.1: Examples of recordings from OpenNEEDS, showing indoor and outdoor scenes and various tasks during gameplay. The green dot represents the 3D gaze vector [12].

The OpenNEEDS dataset covers most of our needs:

- It provides a total of 2,194,865 samples of data.
- It is captured from 44 different participants *(age (years) = 31.7 (SD 10.5); 20 females; 40 right-handed; ipd(mm) = 62.95 (SD 3.62)).*
- It includes two different types of scene (indoor and outdoor).

- It includes numerous different tasks, such as reading, throwing, object-manipulation, drawing, aiming, and shooting.
- It includes continuous data samples, as for each participant they recorded up to 5 minutes of their exploration.

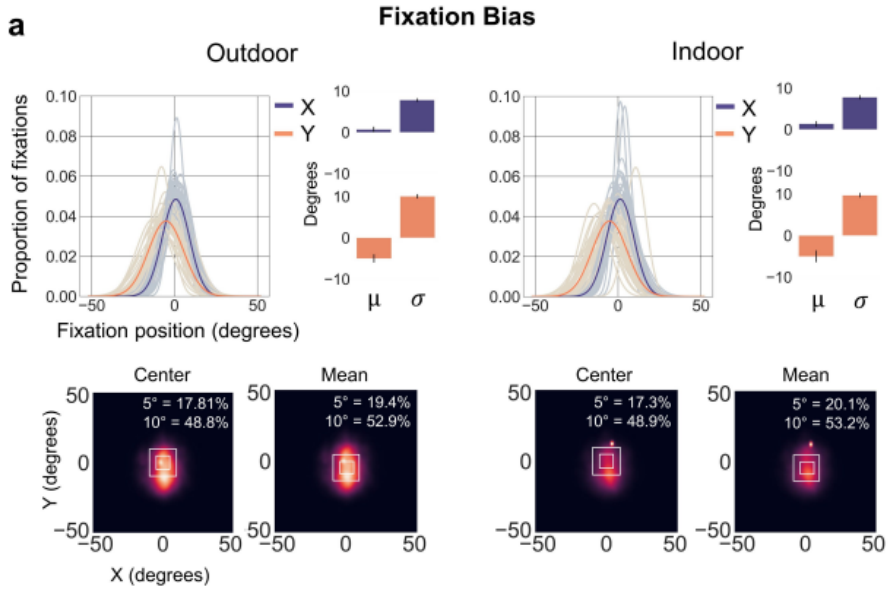


Figure 4.2: Dataset analysis, showing the center and mean biases. Approximately 20% of all fixations lie within 5°eccentricity of the screen center and 50% of all fixations lie within 10°of the fixation mean respectively [12].

The dataset includes a variety of data, from head and hand orientation, to position of interactive objects, to scene frames and gaze data. For this thesis, the following were used:

- **Scene**: The on-screen image presented to the user at each frame is stored as a 3-channel (RGB) image at an 8-bit resolution in sRGB color space, down-sampled to a pixel resolution of 128 x 71.
- **Gaze**: Ground truth 3D gaze vectors in meters are provided for each frame. Gaze X and Y were limited by the FOV (104°), and gaze Z (depth) limits were [0.3,5] meters.

For each sample, there was additional information necessary such as a user ID, a scene ID, and a timestamp.

4.2 Preprocessing

Before training our model, it was necessary to preprocess the dataset to fit the required format.

In the model described in Section 4.3, sequences of past data samples are used as input. Each sample consists of two separate sequences: a sequence of past frames and a sequence of corresponding gaze points. The process of sequence creation must be carefully controlled to ensure that no overlap occurs between different data collection sessions. This means that the data used to create a sample must belong to the same user, the same scene, and form a continuous sequence. The same criteria apply to the corresponding label for each sample, which is a single gaze point (it must also originate from the same session).

To improve the model's performance, the frames are normalized. The gaze points are also normalized and converted into visual angles.

4.2.1 Load Dataset

The dataset, stored in *.parquet* format, is first loaded using the Pandas library and converted into a DataFrame for easy manipulation. The dataset is then cleaned by removing any rows containing NaN values. Following this, the DataFrame is filtered to retain only the columns relevant to our task.

```
print("Reading dataset..............\n")
df= pd.read_parquet(pathDataset)

print("Cleaning dataset..............\n")
#Drop rows with NaN & keep only useful data
df.dropna(inplace=True)

#Reset duplicates indices
duplicated_indices = df.index[df.index.duplicated()]
df.reset_index(drop=True, inplace=True)
```

```
11
12 #Keep useful columns only
13 df = df[['subject_id','time', 'scene_index','images_color',
14          'gaze_x', 'gaze_y', 'gaze_z']]
```

4.2.2 Preprocess Gaze Vectors

4.2.2.1 Transform to Visual Angles

As mentioned earlier, gaze information is provided as a 3D gaze vector (*gazeX, gazeY, gazeZ*). Each vector is transformed into a 2D visual angle (θ, ϕ), using the formula from 3.2. To perform this transformation, the following function was implemented.

```
1 def tranform_to_visual_angle(gazeX, gazeY, gazeZ):
2     theta = np.arctan2(gazeX,gazeZ)
3     phi = np.arctan2(gazeY,gazeZ)
4     theta = np.rad2deg(theta) #turn to degrees
5     phi = np.rad2deg(phi)
6     return theta,phi
```

The function also converts the resulting angles from radians to degrees for easier interpretation.

4.2.2.2 Remove Outliers

Before normalizing the gaze angles, outliers are removed from the data to improve model robustness and performance. Outliers are defined as extreme values that lie significantly outside the typical range of the data. In the context of gaze prediction, outliers might arise due to noise in the data, measurement errors, or rare events that are not representative of normal gaze behaviour. The **Interquartile Range (IQR) Method** is used to detect and remove these outliers.

First, a function was created to detect outliers. The process begins by calculating the first and third quartiles (Q_1 and Q_3 respectively), where Q_1 is the value below which 25% of the data lies, and Q_3 is the value below which 75% of the data lies. The IQR, or the difference between Q_3 and Q_1, is then calculated, representing the range where

the middle 50% of the data is concentrated. Using this IQR, boundaries for detecting outliers are defined:

- **Lower bound:** Any data point below $Q_1 - 1.5 \times IQR$ is considered an outlier.
- **Upper bound:** Any data point above $Q_3 + 1.5 \times IQR$ is also considered an outlier.

This method assumes that most of the data lies within 1.5 times the IQR from the quartiles and flags anything outside this range as an outlier.

```
def calculate_outliers_from_column(column):
    #IQR
    Q1 = np.percentile(column, 25, method='midpoint')
    Q3 = np.percentile(column, 75, method='midpoint')
    IQR = Q3 - Q1
    #Calculate the upper and lower limits
    upper = Q3 + 1.5 * IQR
    lower = Q1 - 1.5 * IQR
```

Thus, outliers are identified as points that fall outside these bounds.

```
    upper_array = np.where(column >= upper)[0]
    lower_array = np.where(column <= lower)[0]
    outliers = np.hstack((upper_array,lower_array))
```

Then, another function applies this detection logic to both horizontal ($angleX$) and vertical angles ($angleY$). After detecting outliers in each direction, the outlier indices are combined, and duplicated outliers are removed using *np.unique()*.

```
def remove_outliers(dtframe):
    gazex = dtframe['angleX']
    gazey = dtframe['angleY']
    outliersX = calculate_outliers_from_column(gazex)
    outliersY = calculate_outliers_from_column(gazey)

    full_outliers = np.hstack((outliersX, outliersY))
    full_outliers = np.unique(full_outliers)
    print("Full number of  unique outliers:", full_outliers.shape[0])
```

Finally, the rows containing these outliers are dropped from the DataFrame:

```
    dtframe.drop(dtframe.index[full_outliers], inplace=True)
    return dtframe
```

4.2.2.3 Gaze Normalization

Once the gaze vectors are transformed into 2D visual angles (θ, ϕ) and the outliers are removed, normalization is performed to scale the angles into a standard range [0,1]. This is important because many machine learning models work better when the input features are normalized. A simple min-max normalization formula was implemented, following 3.3.

```
def normalize_visual_angle(column):
    min_val = column.min()
    max_val = column.max()
    return (column - min_val) / (max_val - min_val)
```

Before normalization is performed, the minimum and maximum values of the gaze angles are computed. These values are necessary not only for normalization but also for **denormalization** later when making predictions, allowing the predicted values to be scaled back to their original range.

4.2.3 Preprocess Frames

For the frames, a simple function to handle both loading and normalizing the image data was implemented. This preprocessing is essential to ensure that the images are in a format suitable for feeding into a neural network.

The OpenCV library is used to read each image from a given file path. This function reads the image and stores it as a multi-dimensional NumPy array with pixel values in the range [0, 255]. Each pixel is represented by three values corresponding to the RGB color channels. Since the original pixel values range from 0 to 255, they are normalized by dividing by 255, which scales all pixel values to fall within the range [0, 1] and helps

the model converge faster during training. If an error occurs (such as a file not being found or an image being corrupted), the function returns None and prints a message.

```
def load_norm_images(image_path):
  try:
    img = cv2.imread(image_path)
    img = np.asarray(img) / 255.0
    return img
  except Exception as e:
    print(f"Error loading image {image_path}: {e}")
    return None
```

Due to the vast size of the dataset, loading and processing all images simultaneously is not feasible. Instead, a more efficient approach is adopted by loading and normalizing images on demand. This strategy will be further elaborated in Section 4.2.4, where the process of calling the image loading and normalization function as needed is detailed.

In the dataset, the image file names are stored in the DataFrame, while the actual images are located in a separate zipped folder. After this folder is unzipped, each image name in the DataFrame needs to be linked to its correct file path so the images can be easily accessed during preprocessing.

A function is implemented to append the folder path to the image name, creating the full path by concatenating the folder path with the image file name and ensuring that the resulting path is correctly formatted for the operating system.

```
def correct_path(image_name):
    full_path = os.path.join(pathToImageFolder, image_name)
    return full_path
```

The function is then applied to the corresponding column of the dataframe which contains the image file names.

```
print("Getting correct image paths.........")
df['images_color'] = df['images_color'].apply(correct_path)
```

4.2.4 Create Sequences

At this stage, all necessary tools are in place, including preprocessed and normalized gaze data and image frames, to generate sequential samples for model training.

The goal is to create sequences of data, where each sequence consists of consecutive frames and their corresponding gaze points. For instance, to create sequences of size 3, the following components are taken:

- Frames: I_0, I_1, I_2
- Gaze Points: GP_0, GP_1, GP_2

The label for this sequence will be the gaze point corresponding to the next frame in the sequence, GP_3, but without the frame I_3. This setup ensures that the model learns to predict the future gaze point based on the current sequence of frames and gaze information, without needing access to the future frame.

This approach allows the model to effectively learn gaze behaviour patterns over time, predicting where the user will look next based on the past few frames.

Importantly, for each sequence, it must be ensured that the data represents an uninterrupted sequence. This requires that the data originates from the same user and the same scene, with frames in consecutive order. Ensuring continuity is essential for effective model training, as any interruption (e.g., switching users or scenes) could disrupt the temporal patterns the model is intended to learn.

With these requirements in place, the process of creating sequential samples will now be broken down. This process involves loading and normalizing images in batches, splitting them into sequences, and validating each sequence before saving it in the required format for model training.

4.2.4.1 Load Images in Batches

The first step involves loading images in manageable batches, rather than processing the entire dataset at once. This is crucial given the dataset's size. Loading all the images at once would lead to excessive memory usage, overwhelming the available RAM. To manage this, a Python generator to load the images in smaller batches is utilized.

A generator in Python is a special type of iterable that allows us to yield values one at a time rather than loading everything into memory at once. Instead of returning all the

images immediately, a generator will load and process each batch, allowing us to iterate through the dataset without consuming too much memory at once.

```
def load_images_in_batches(dtframe, batch_size, limit, starting_point):
    #calculate  total batches
    num_batches = int(np.ceil(len(dtframe) / batch_size))
    #start from the correct batch
    start_batch = int(np.floor(starting_point / batch_size))
    processed_count = starting_point
    for i in range(start_batch, num_batches):
        batch = df.iloc[i * batch_size: (i+1) * batch_size]
        indices_list = []

        for idx, row in batch.iterrows():
            image = load_norm_images(row['images_color'])
            #In case of issue loading image, set to NaN and drop entire row
            if image is None:
                dtframe.iloc[idx,3] = np.nan
                dtframe.dropna(inplace = True)

        processed_count+=1
        indices_list.append(idx)
        if processed_count >= limit:
            break
        yield np.array(indices_list)    #output batch!
        if processed_count >= limit:
            break
```

The generator processes a specific number of images (defined by batch size). It also incorporates a starting point and a limit, allowing us to control which portion of the dataset is loaded, in case we need to manage this process manually. Moreover, the generator handles any errors in image loading by marking and dropping the problematic rows. The function yields the indices of successfully loaded images in each batch for further processing.

4.2.4.2 Split Indices into Sequences

Once the indices of the images have been obtained, they are split into sequences based on the specified sequence length:

```
def split_list(my_list, seq_length):
    true_seq = seq_length + 1
    #create sublists
    sublists = [my_list[i:i + true_seq] for i in range(0, len(my_list), true_seq)]
    if len(sublists[-1]) < true_seq:
        sublists.pop()  # remove incomplete sequences
    return sublists
```

This function splits the list of image indices into sublists of length (sequence length + 1). The additional 1 is because the gaze point of the next frame in needed as the label. If the last sublist is shorter than the required length, it's discarded, to ensure full sequences.

4.2.4.3 Create Sequences

For each set of image indices generated, continuity is checked, and input-output pairs are created. The following loop is responsible for generating sequences by ensuring that all frames in the sequence belong to the same user and the same scene. It takes a large batch of images, splits it into lists of potential sequences, and validates each list to confirm whether it forms a valid sequence. For each iteration, three key components are created:

- X1: the sequence of frames.
- X2: the sequence of gaze points.
- Y: the gaze point corresponding to the next frame, which will be used as the label.

```
for indices in image_generator: #split batch to potential sequences
   split_indices = split_list(indices, sequence_length)
   #for each, init lists and necessary counters
   for miniList in split_indices:
      starting_index = miniList[0]
      X1, X2, Y = [], [], []
      row = df.iloc[starting_index]
      success = element_counter = 0
```

For each potential sequence, every element of the list is compared with the first element to ensure that all frames originate from the same user and scene. If the sequence

is valid, the data is stored in the appropriate components. Once the sequence is complete, it is saved.

```
      #compare next element with original for sequence validation
      for element in miniList:
         nextRow = df.iloc[element]
         if (nextRow['subject_id'] == row['subject_id'] and
             nextRow['scene_index'] == row['scene_index']):
            success += 1
            #reach here if element is suitable
            #check if its part of the sequence or the label
            if element_counter <= sequence_length - 1:
              X1.append(load_norm_images(nextRow['images_color']))  #image data
              X2.append(nextRow[['angleXNorm', 'angleYNorm']].values)  #gaze data
            else:
               Y.append(nextRow[['angleXNorm', 'angleYNorm']].values)  #GP label
         else:
            print("Rejected due to element: ", element)
            break
         if success == sequence_length + 1:                #sequence is ready!
            total_sequences += 1
            append_to_hdf5(filename, X1, 'X1_dataset', (sequence_length,h,w,c))
            append_to_hdf5(filename, X2, 'X2_dataset', (sequence_length,2))
            append_to_hdf5(filename, Y,  'Y_dataset',  (1,2))
         element_counter += 1
print("\nTotal sequences created:", total_sequences)
```

4.2.4.4 Store Final Dataset

To store the final dataset, the HDF5 file format was chosen (see Section 3.4.2). HDF5 allows for efficient storage and handling of large datasets, offering flexibility in structuring data hierarchically. It also enables the creation of datasets that can be dynamically resized, which is essential when dealing with large data processed in batches.

In the implementation, the HDF5 file is initialized by defining empty datasets with appropriate dimensions for X1 (frames), X2 (gaze points), and Y (label). The datasets are created with flexible shapes, allowing them to grow as more sequences are added.

For example:

```
with h5py.File(filename, 'w') as f:
   f.create_dataset('X1_dataset', shape=(0, sequence_length, h, w, c),
                    maxshape=(None, sequence_length, h, w, c), chunks=True)
```

The first dimension represents the number of samples (sequences), the second represents the number of frames in each sequence (sequence length), and the third, fourth, and fifth dimensions represent the height, width, and channels (color depth) of each image.

Each time a new sequence is created, it is appended to the corresponding dataset with the following function. It checks if the dataset exists; if not, it creates it. If the dataset exists, it resizes it to accommodate the new data and appends the new sequence:

```
def append_to_hdf5(file_name, data, dataset_name, shape):
    with h5py.File(file_name, 'a') as f:
        if dataset_name not in f:
            # Create dataset if it doesn't exist
            f.create_dataset(dataset_name, data=data, maxshape=(None, *shape),
                                                                chunks=True)
        else:
            # Resize and append data to the existing dataset
            dataset = f[dataset_name]
            dataset.resize(dataset.shape[0] + 1, axis=0)
            dataset[-1] = data
```

To sum up, the final dataset is now stored in HDF5 files, with sequences of frames and gaze points properly formatted and ready to be used as input for the neural network model.

4.2.5 Training, Validation, Testing Dataset Generators

In our implementation, data is prepared for training, validation, and testing in a memory-efficient manner. Since the dataset is too large to be loaded entirely into RAM, Python generators are again used to load data in smaller batches. This approach aligns with batch processing in neural network training, where data is incrementally loaded in manageable chunks, allowing the model to process one batch at a time. By applying this method with HDF5-stored data, memory overload is avoided, ensuring efficient and scalable training.

First, the dataset had to be split into training, validation, and testing sets, with a ratio of 80% for training, 10% for validation, and 10% for testing. Since the dataset is divided across four separate HDF5 files, a function was created to calculate and identify the appropriate indices for this split.

```
#Function to separate indices of dataset to training, validation & testing
#Returns: final index for each and num of total samples
def split_train_val_test(filenames):
    file_sizes = []
    total_samples = 0
    # Calculate size of each file and add to get total number of samples
    for filename in filenames:
        with h5py.File(filename, 'r') as f:
            file_size = len(f['X1_dataset'])
            file_sizes.append(file_size)
            total_samples += file_size
            print(f"Size: {file_size} for file:{filename}")
    print("Total samples: ", total_samples)
    train = total_samples * 0.8
    val = train + 0.1 * total_samples
    test = total_samples
    return int(train), int(val), int(test), total_samples
```

This function determines the total number of samples by iterating over each file to retrieve the size of each dataset within. Using the total sample count, the function calculates the indices corresponding to the training (80%), validation (10%), and testing (10%) portions, returning these as integer indices for later use in data generation.

Then, the generator was implemented. The generator function yields batches of data for model training, validation, and testing, following a series of steps.

It first calculates the sizes of each dataset within the files to understand how many samples each contains. Next, the function determines the appropriate file and index based on current_position to ensure that all samples are loaded from the correct location, as shown below.

```
    # Start from startPoint
    current_position = startPoint
    while True:
```

```
        # Find the current file and local position based on `current_position`
        file_index = 0
        cumulative_position = 0
        added_file_sizes_position = 0
        # Determine the correct file index and local position
        while (file_index < len(file_sizes)
          and current_position>= added_file_sizes_position + file_sizes[file_index]):
            added_file_sizes_position += file_sizes[file_index]
            file_index += 1
        if file_index == len(file_sizes):
            # If we've reached beyond the last file, reset
            current_position = startPoint
            continue
        # Local position within the current file
        local_position = current_position - added_file_sizes_position
        filename = filenames[file_index]
        #print(f"Using file with index {file_index}: {filename}")
```

Within each selected file, the function fetches batches of X1dataset, X2dataset, and Ydataset from the local position. It yields these batches, which consist of the frame sequences (X1batch), gaze points (X2batch), and the label (YBatch).

```
        with h5py.File(filename, 'r') as f:
            X1_dataset = f['X1_dataset']
            X2_dataset = f['X2_dataset']
            Y_dataset = f['Y_dataset']
            # Calculate the number of samples in the current file
            num_samples = file_sizes[file_index]
            # Continue from the current position within this file
            while local_position < num_samples and current_position < endPoint:
                # Calculate batch indices
                batch_end = min(local_position + batchsize, num_samples)
                batch_indices = np.arange(local_position, batch_end)
                # Create batches
                X1batch = X1_dataset[batch_indices]
                X2batch = X2_dataset[batch_indices]
                YBatch = Y_dataset[batch_indices]
                # Yield the current batches
                yield (X1batch, X2batch), YBatch
```

```
                # Update positions
                local_position += batchsize
                current_position += batchsize
            # Move to the next file
            local_position = 0
        # If we've exhausted all files, reset the position to startPoint
        if current_position >= endPoint:
            current_position = startPoint
```

When the dataset portion is fully loaded, it resets the current position to allow another full pass if needed.

To use the generators, they are initialized, with the appropriate indices calculated by the *split_train_val_test* function. Then, they are given as input to the neural network.

```
paths = [final_path1, final_path2,final_path3,final_path4]
#Parameters settings
batchsize = 64
limit = None
train, val, test, tot_samples = split_train_val_test(paths)
#init generators
TrainingDatasetGenerator = datasetGenerator(paths, batchsize, 0, train, limit)
ValidationDatasetGenerator = datasetGenerator(paths, batchsize, train,val, limit)
TestingDatasetGenerator = datasetGenerator(paths, batchsize, val, test, limit)
```

The total number of samples (size of dataset) is 172071 samples. 137656 are used as the training data, 17208 for validation and 17208 for testing data.

4.3 Model Architecture

In this section the system architecture will be explored. A system overview will be presented, explaining the reasons behind our choices concerning the architecture. Then, each subsystem will be described in detail.

4.3.1 System Overview

Our research has revealed that temporal information is crucial for predicting gaze behaviour in task-oriented VR environments. When users explore their surroundings, their

gaze is typically not erratic; rather, there is continuity and consistency in their gaze position sequences based on the tasks they are currently performing. For this continuity to break, a stimulus must appear that draws their attention. Additionally, there is a delay between the appearance of a stimulus and the user's response to it, due to human reaction time.

More and more models are currently being implemented, as discussed in Chapter 2, with some incorporating the concept of temporal continuity by utilizing sequences of past object positions, past head velocities, or past gaze points. However, these models often emphasize the importance of static scene information for predictions, which is why they employ saliency encoder modules to process frames. Most of these models, such as DGaze [11], require multiple data inputs to make their predictions.

We believe there is a need to explore the concept of temporal continuity in depth and investigate whether it is sufficient on its own to predict gaze. Thus, we present the architecture of the system proposed.

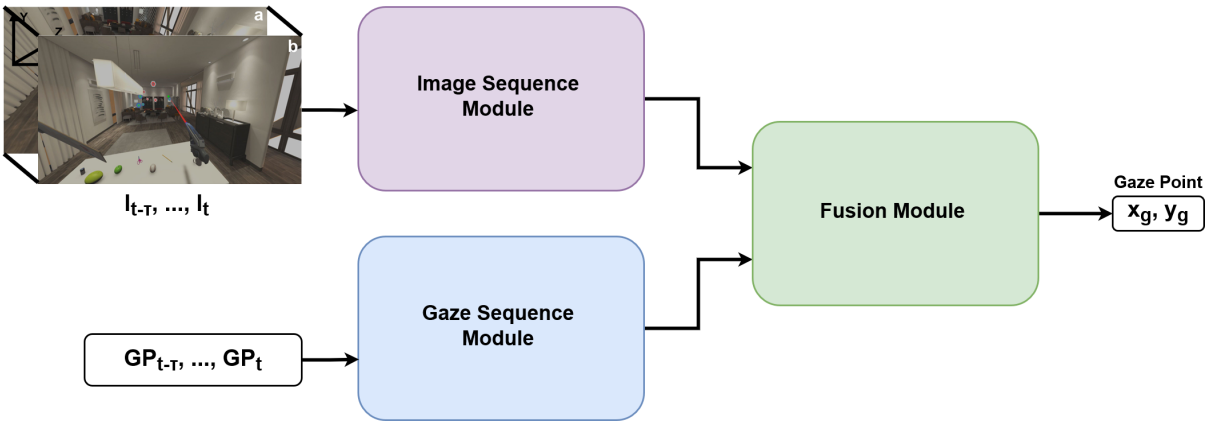


Figure 4.3: Architecture of the proposed model

The model is composed of three modules: image sequence module, gaze sequence module, and gaze fusion module. The first one is used to learn the temporal motion features among consecutive frames. The second is utilized to learn the temporal patterns of gaze. Finally, the third module fuses the outcomes and makes a single gaze prediction.

4.3.2 Image Sequence Module (ISM)

Since temporal information plays a key role in dynamic environments, using each frame as a static image for prediction seems unreasonable. However, some knowledge of the scene is necessary to identify stimuli or salient parts of the frame that inevitably draw attention. LSTM networks possess strong long-term memory capabilities, making them suitable for our needs. ConvLSTM layers enhance this by replacing the dot product operation with convolution, allowing us to process a sequence of frames to memorize temporal information. Moreover, ConvLSTM does not focus on the features of each individual frame; instead, it learns the correlations of features across frames, capturing the dynamic relationships within the sequence.

For all these reasons, the ISM takes a sequence of frames as input and learns the temporal motion features among consecutive frames.

At time t, a sequence of frames $I_{t-\tau}, I_{t-\tau+1}, ..., I_t$ is fed into the network. The sequence consists of 10 frames ($\approx 100ms$).

We employ 5 ConvLSTM2D layers with 16 filters, a kernel size of 3×3, same padding, and ReLU as the activation function. The return sequences flag is set to True for the first 4 layers, allowing each subsequent layer to receive a full ten-dimensional sequence as input. For the last layer, the flag is set to False. After each of the first four ConvLSTM layers, a MaxPooling3D layer is used to reduce the dimensions of the feature maps. They help reduce the number of parameters to learn and the amount of computation performed in the network. They also assist in making the model more robust to variations in the position of the features in the input images.

The output is then Flattened, followed by 4 Fully Connected Layers to encode the information, with sizes of 64, 32, 32, and 16 respectively. A Dropout Layer with a dropout rate of 0.5 is included to prevent overfitting. The system's input is a 4D tensor with shape (sequence length, height, width, channels), specifically (10, 71, 128, 3).

4.3.3 Gaze Sequence Module (GSM)

At time t, a sequence of gaze points $GP_{t-\tau}, GP_{t-\tau+1}, ..., GP_t$ is fed into the network. This sequence also consists of 10 gaze points ($\approx 100ms$).

Four LSTM layers are utilized, with 20 units in the first 3 and 10 in the last one. ReLU is used as the activation function, with the return sequences flag set to True for

the first three layers and False for the last one. The output is then Flattened, followed by 2 Fully Connected Layers to encode the information, with sizes of 32 and 16 respectively. A Dropout Layer with a dropout rate of 0.5 is also included to prevent overfitting. The system's input is a 2D tensor with the shape (timesteps, gaze vector), specifically (10, 2).

4.3.4 Fusion Module (FM)

In the Fusion Module, the outputs from the Image Sequence Module and the Gaze Sequence Module are merged using a Maximum operation. This operation combines the features from both modules, allowing for the integration of temporal frame information and past gaze point data.

Following the merge, a fully connected layer of size 16 and ReLU as the activation function is employed to further process the combined features. To prevent overfitting, a Dropout Layer with a dropout rate of 0.5 is applied after this hidden layer.

The final output layer consists of a fully connected layer of size 2, with sigmoid as the activation function. This layer generates the final gaze prediction as a two-dimensional output (x_g, y_g), with values ranging from 0 to 1.

4.4 Training

In this section, we outline the environment and configurations used for training the deep learning model designed for gaze prediction. We present the training details, including the loss function, optimizer, epochs, callbacks etc.

4.4.1 Environment Setup

The model was trained on Google Colab using the NVIDIA L4 Tensor Core GPU, enabling efficient processing and faster training times. This powerful hardware setup supports the demands of deep learning tasks and experimentation.

The implementation was developed using Python 3.10.12. The model framework is based on TensorFlow 2.17.0 with Keras 3.4.1.

4.4.2 Training Configuration and Optimization Settings

To train the model, Mean Absolute Error (MAE) is utilized as the loss function. The Adam optimizer is employed with an initial learning rate of 0.001. The training process uses a batch size of 64 and is conducted over 10 epochs.

4.4.3 Callback Mechanisms

Two different callbacks are implemented to enhance the training process. First, the ReduceLROnPlateau callback is employed to monitor the loss and adjust the learning rate. This callback reduces the learning rate when the loss ceases to improve, helping the model adapt to the training dynamics.

Second, the EarlyStopping callback is utilized to monitor the loss and terminate training when there is no significant improvement over a specified number of epochs. This callback helps identify the point at which the model stabilizes, preventing unnecessary training and potential overfitting.

```
reduce_lr = keras.callbacks.ReduceLROnPlateau(monitor = "val_loss", patience = 5,
                                                                    verbose = 1)
early_stopping = keras.callbacks.EarlyStopping(monitor = "val_loss", patience = 10,
                                                                     verbose = 1)
```

Chapter 5

Evaluation

In this chapter, we will present the evaluation of our system. We will explain the baselines and metrics used, along with their implementation. Finally, we will provide a detailed discussion of the results obtained from our system.

5.1 Baselines and Evaluation Metrics

5.1.1 Baselines

For the task of real-time gaze prediction, following the works of [50, 11, 64], as well as the conclusions we made from the dataset analysis, we use the following baselines for evaluation:

- First of all, the **center baseline (0°, 0°)**. This baseline is meaningful since the dataset revealed a center bias.
- the **mean baseline**, which in our dataset is **(1.189°, -5.259°)**. This baseline is also meaningful since a large percentage of fixations lie within 10°of the mean.

5.1.2 Evaluation Metrics

We use the **angular distance** between the predicted gaze position and the ground truth gaze position as the evaluation metric.

Assuming predictions with visual angles (θ_i, ϕ_i) and a baseline ($\theta_{baseline}$, $\phi_{baseline}$), we can calculate the difference in each plane (horizontal, vertical):

$$\Delta\theta_i = \theta_i - \theta_{baseline}$$

$$\Delta\phi_i = \phi_i - \phi_{baseline}$$

Then, the overall angular difference (magnitude) between the gaze points and the baselines can be calculated with the Euclidean distance in the angular space:

$$\Delta angle_i = \sqrt{(\Delta\theta_i)^2 + (\Delta\phi_i)^2}$$

To further assess the reliability of our model's improvement over the baseline, the **Standard Error of the Mean (SEM)** was calculated, as well as the **Cumulative Distribution Function (CDF)**. In addition, for many applications like gaze-contingent rendering, the gaze region plays a more important role than a single gaze position. Thus, we use the **Recall Rate**, to determine the proportion of the overlapped region at the ground truth region, as an evaluation metric. The higher the recall rate, the better the performance.

Finally, the **runtime performance** is evaluated.

5.2 Model Evaluation Implementation

Here, we present the necessary code for the evaluation of the model. First, two functions were implemented to calculate the **Average Angular Error**:

```
import numpy as np
#calculate angular error between two points
def angular_difference(theta1, phi1, theta2, phi2):
    delta_theta = theta1 - theta2
    delta_phi = phi1 - phi2
    return delta_theta, delta_phi


#Calculate final angular error for all points
```

```
def AngularError(data1, data2):
  delta_thetas, delta_phis = angular_difference(data1[:,0], data1[:,1],
                                                data2[:,0], data2[:,1])
  #Overall angular error (Euclidean distance in angular space)
  full_errors = np.sqrt(delta_thetas**2 + delta_phis**2)
  #mean angular
  mean_error = np.mean(full_errors)
  return full_errors, mean_error
```

Additionally, it was necessary to perform **reverse normalization** on the predictions to ensure accurate evaluation. By using the true maximum and minimum values from the original data, we can transform the predictions from the range [0,1] back to the original dataset's range.

```
def denormalize(data,true_min, true_max):
  denormalized = data * (true_max - true_min) + true_min
  return denormalized

def denormalize_full(data, true_minX, true_maxX, true_minY, true_maxY): #data = (x,y)
  Xgazedenorm = np.array(denormalize(data[:,0], true_minX, true_maxX))
  Ygazedenorm = np.array(denormalize(data[:,1], true_minY, true_maxY))
  final_denormalized = np.column_stack((Xgazedenorm,Ygazedenorm))
  return final_denormalized     #data with shape (x,y)
```

We also define the two **baselines**:

```
#MEAN baseline                 #calculated from groundtruth data(1.1891, -5.2595)
mean_theta = 1.1891425755626226
mean_phi = -5.259580611647258
mean_baseline = np.full((1,2), (mean_theta, mean_phi))
print(f"Mean Baseline: {mean_baseline}")

#CENTER baseline               #(0,0)
center_theta = center_phi = 0
center_baseline = np.full((1,2),0)
print(f"Center baseline: {center_baseline}")
```

Then, similarly to 4.2.5, we parse the dataset file list to locate the testing dataset and

make predictions in batches to avoid filling up the RAM. After determining the correct file index and local position, for each batch in the file, we use the model to make predictions. Then, denormalization is performed, and for each batch, the Angular Distance (Error) is calculated for the predictions as well as the baselines.

```
batch_predictions = model.predict((X1batch,X2batch))        #predict

#reverse normalization
final_batch_pred = denormalize_full(batch_predictions, Xmin, Xmax, Ymin, Ymax)
final_batch_GT = denormalize_full(Ybatch, Xmin, Xmax, Ymin, Ymax) #groundtruth
#calculate prediction error
batch_errors, batch_average = AngularError(final_batch_GT, final_batch_pred)
full_errors.append(batch_errors)
#calculate mean baseline error
mean_batch_error, mean_batch_average = AngularError(final_batch_GT, mean_baseline)
full_error_mean.append(mean_batch_error)
#calculate center baseline error
center_batch_error, mean_batch_average= AngularError(final_batch_GT, center_baseline)
full_error_center.append(center_batch_error)
# Update positions
local_position += batch_size
current_position += batch_size
```

To calculate the **SEM** for our model, we first determine the mean and the standard deviation. Using these values, we calculate the SEM and the confidence interval (CI). Similarly, we perform the same calculations for the baselines.

```
# Calculate means
mean_model = np.mean(final_prediction_error)
# Calculate standard deviations
std_model = np.std(final_prediction_error, ddof=1)
# Calculate SEMs
sem_model = std_model / np.sqrt(len(final_prediction_error))
# Calculate 95% confidence intervals
ci_model = [mean_model - 1.96 * sem_model, mean_model + 1.96 * sem_model]
```

To calculate the **CDF** of our model, we first divide the error data into 10 bins using a histogram. The CDF is then computed by taking the cumulative sum of the histogram

count and multiplying it by the width of the bins, which represents the cumulative probability up to each bin. Similarly, we perform the same calculations for the baselines.

```
# Getting data of the histogram
count_predictions, bins_count_predictions= np.histogram(final_prediction_error,
                                                        bins=10, density = True)
#calculate the CDF
cdf_predictions = np.cumsum(count_predictions) * np.diff(bins_count_predictions)
```

For the **recall rate**, a function to calculate the Euclidean Distance between two sets of points is defined.

```
# Function to calculate Euclidean distance (in degrees) between two sets of points
def calculate_distance(gt, pred):
    return np.sqrt((pred[:, 0] - gt[:, 0])**2 + (pred[:, 1] - gt[:, 1])**2)
```

The necessary parameters are then initialized:

```
# Parameters for region radius
ground_truth_radius = 15  # degrees for ground truth region
predicted_radius = 15     # degrees for predicted region (both model and baseline)
#Initialize lists for recall counting
model_hits, mean_hits, center_hits = 0
```

Following a similar logic as previously, the hits for the recall rate are calculated.

```
  #Calculate distances for model predictions, mean baseline, and center baseline
  distances_model = calculate_distance(final_batch_GT,  final_batch_pred)
  distances_mean = calculate_distance(final_batch_GT, mean_baseline)
  distances_center = calculate_distance(final_batch_GT, center_baseline)

  #Count hits based on the ground truth and prediction radii
  model_hits += np.sum(distances_model <= ground_truth_radius)
  mean_hits += np.sum(distances_mean <= predicted_radius)
  center_hits += np.sum(distances_center <= predicted_radius)
  #Update positions
```

```
    local_position += batch_size
    current_position += batch_size

# Calculate recall rates
model_recall_rate = model_hits / total_samples
mean_recall_rate = mean_hits / total_samples
center_recall_rate = center_hits / total_samples
```

Finally, to evaluate the **runtime performance**, we simply measure the time it takes to make a single prediction. The Keras library provides utilities to accurately measure the prediction time, allowing us to evaluate the model's latency.

5.3 Model Evaluation Results

We present the results of our evaluation.

5.3.1 Performance Evaluation

To compare the Average Angular Error, we chose to visualise the results using a boxplot. This method provides a detailed representation of the data, highlighting the median, variability, and potential outliers, allowing for multiple conclusions to be drawn.

5.3.1.1 Median Error

The **median**, displayed as the line inside each box, represents the middle value of the angular error distribution for each method (Center Baseline, Mean Baseline, Predictions).

The **center** baseline has a median of 13.15°, suggesting that although the dataset revealed a center bias, it is not a reliable form of prediction. Similarly, the **mean** baseline has a median of 11.95°. The median of our **model**, however, is noticeably lower than the previous ones, at 4.41°. This shows that the model's **central tendency** is much closer to the true gaze point, achieving a 66.43% improvement over the center baseline and a 63.08% over the mean baseline, calculated with the following formula:

$$(err_{baseline} - err_{model})/err_{baseline}$$

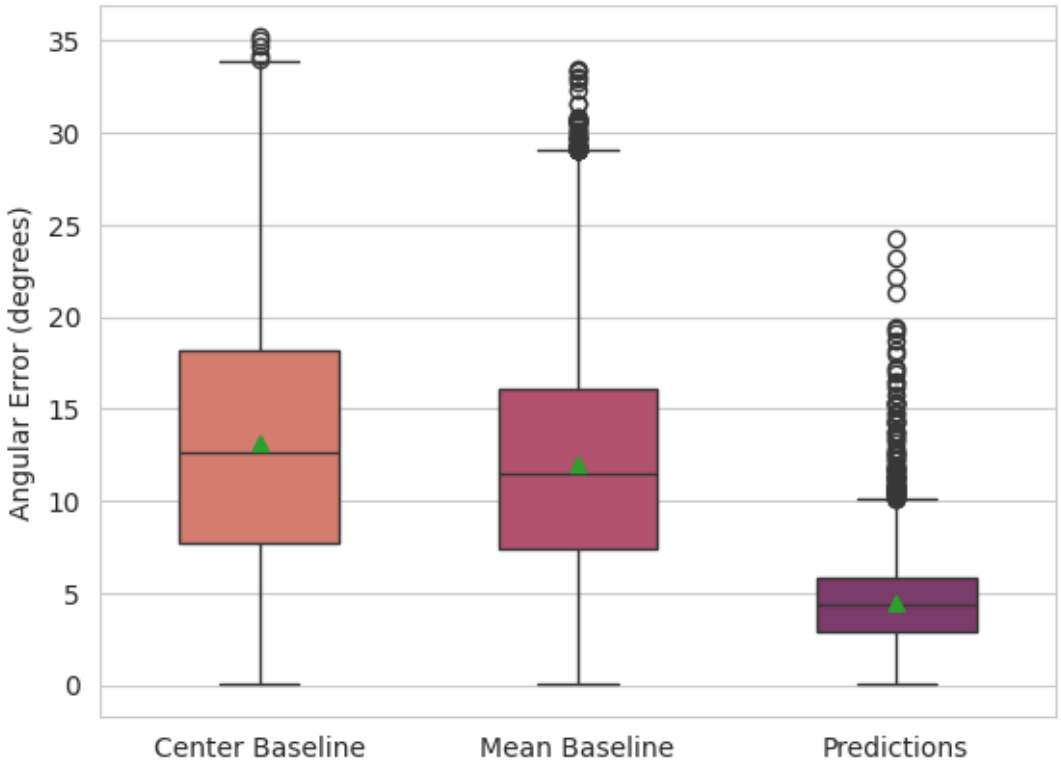


Figure 5.1: Angular Error comparison between our model and the baselines

5.3.1.2 Interquartile Range (IQR)

The **IQR** is represented by the height of each box and includes the middle 50% of the data, from the 25th percentile (bottom of the box) to the 75th percentile (top of the box).

The model's predictions have the **smallest** IQR, indicating that the errors produced by the model are more tightly clustered around the median. A narrow IQR here reflects that the model's predictions are more consistent, with fewer large deviations, whereas the wider IQRs for the baselines suggest a greater variance in their errors.

5.3.1.3 Whiskers

The **whiskers** extend from the edges of each box to capture the range of non-outlier values in the data.

For our model, the whiskers are notably **shorter** than those for the baselines, meaning that the bulk of errors for our model do not stray far from the IQR. This indicates that, beyond just the central 50% of the data, even the more extreme (but still typical) errors in our model's predictions tend to be relatively small. In contrast, the longer whiskers in the baselines reflect a broader spread of errors, which can signal less reliability across different samples.

5.3.1.4 Outliers

Outliers, represented as individual dots outside the whiskers, indicate instances where errors deviate significantly from the central values.

While our model has some outliers, their values are generally lower compared to the outliers for the baselines, which reach as high as 30-35°. This tells us that when our model does make larger errors, they tend to be smaller in magnitude than the largest errors produced by the baselines. This can be particularly important in gaze-contingent applications where high-error predictions may have noticeable, undesirable effects.

5.3.1.5 Performance Relative to Baselines

The boxplots for the **Mean Baseline** and **Center Baseline** not only have higher medians but also much wider ranges of errors, both within the IQR and beyond. This indicates that these baselines are less effective and more erratic, leading to a higher likelihood of making large errors.

By contrast, our model's boxplot shows a **lower median** and a **narrower distribution**, demonstrating that it is not only more accurate but also more dependable. This consistency is essential for applications requiring a stable gaze prediction, as it implies that the model is unlikely to produce extreme errors.

The smaller and more stable error distribution in our model's predictions suggests that it can handle different gaze scenarios better than the baselines, making it a more robust choice. The high error variability in the baselines could lead to unpredictable performance, which is less desirable for real-time applications.

5.3.1.6 Standard Error of the Mean (SEM)

To further assess the reliability of our model's improvement over the baseline, the **Standard Error of the Mean (SEM)** was calculated. The SEM measures the precision of the sample mean—indicating how much the observed mean error would vary if we repeated the experiment multiple times. A smaller SEM implies a more stable and accurate estimate of the mean error. Using the SEM, we constructed a **95% confidence interval (CI)** around the model's mean error. If the results show no overlap between the CIs of the means, that would imply that the differences between the means are statistically significant ($\alpha = 0.05$).

The SEM can be calculated with the following formula:

$$SEM = \frac{\sigma}{\sqrt{N}}$$

where σ is the standard deviation of the sample and N is the sample size.

For a 95% confidence level, the interval around the mean (M) is calculated as:

$$CI = [M - 1.96 \times SEM, M + 1.96 \times SEM]$$

where M is the mean of the prediction errors and 1.96 is the z-score for a 95% confidence level.

Model	Mean Error (degrees)	95% Confidence Interval	SEM
Ours	4.41	[4.38, 4.45]	0.0158
Mean Baseline	11.96	[11.93, 11.99]	0.0158
Center Baseline	13.15	[13.12, 13.19]	0.0158

Table 5.1: SEM

The results, as shown at Table 5.1, are clear and indicate a robust statistical difference between the model and the baselines. First, it is worth noting that the consistently low SEM for all the models suggest that the mean error estimates are stable and precise. The CIs do not overlap, which means that the differences are statistically significant, strengthening the conclusion that our model's lower mean error is unlikely to be due to chance alone. With a precise mean error (low SEM) and a significantly lower error rate than the baselines (non-overlapping SIs), we can conclude that our model's performance is both better and reliably so.

5.3.1.7 Cumulative Distribution Function (CDF)

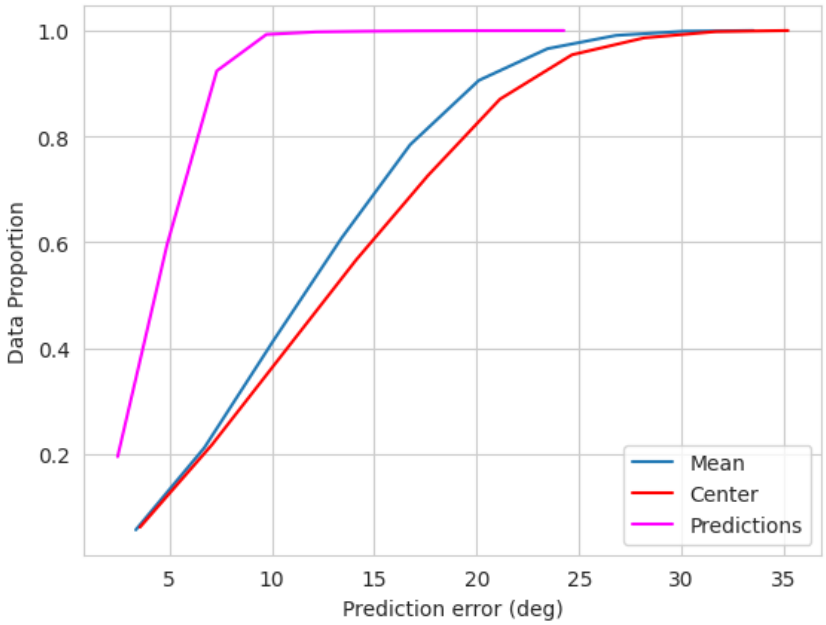


Figure 5.2: Cumulative Distribution Function of the prediction errors

The **CDF** of a real-valued random variable X is the function given by:

$$F_X(x) = P(X \leq x)$$

In our example, this means that for a given error threshold on the x-axis, the y-axis shows the proportion of samples with an error less than or equal to that threshold.

The **Predictions** curve rises steeply, reaching a CDF of 1 very quickly. This suggests that most predictions have relatively low error values, as the probability reaches 1 around an error of 10. The **Mean** and **Center** curves take a more gradual path and reach a CDF of 1 around an error of 30 or so. This implies that a larger portion of their errors are higher compared to our model.

5.3.2 Recall Rate

We also calculate the recall rate of our model and the baselines, as one of the purposes of our model is to be used in gaze-contingent rendering pipelines. The radius is set to 15°,

centered at the ground truth gaze point, as used in foveated rendering [30], and following the work of [11].

	Center	Mean	Ours
Mean Recall Rate	60.8%	69.46%	99.87%

Table 5.2: Recall rates of our model and the baselines

As shown above, our model outperforms the baselines by a significant margin, achieving a recall rate of 99.87%, which is more than adequate for practical applications.

5.3.3 Predictions vs. Groundtruth Data

We also compare the distribution of our neural network's predictions against the ground truth data, using a density heatmap to visualise the differences in the distributions.

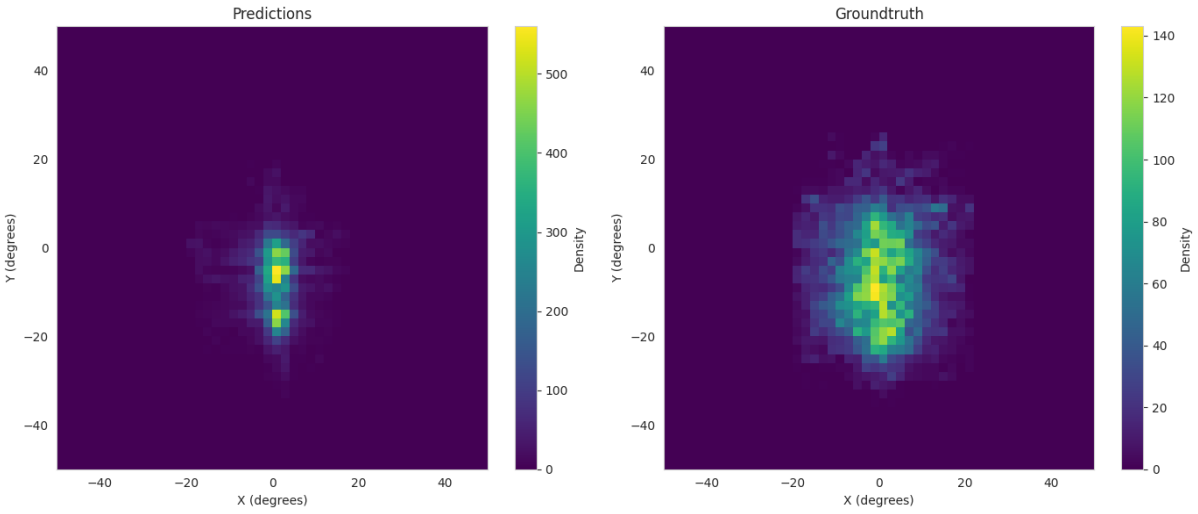


Figure 5.3: Heatmap Comparison of Predictions and Ground Truth Densities

As shown in Figure 5.3, the predictions present a central alignment with high densities concentrated near (0, 0). This suggests that the model might be biased toward central values or overly confident in predicting certain regions. In contrast, the ground truth data is more evenly spread across the plot, with lower peak densities. This indicates that the true data distribution is broader and potentially "noisier" than the model assumes.

Nevertheless, the model demonstrates a strong ability to capture the high-density regions of the ground truth distribution, particularly near the central areas. This suggests that it has effectively learned the core structure of the data. However, the model tends to make "safer" predictions, avoiding extremes and favouring values closer to the center. To improve further, the model could better account for the broader variability and lower-density regions present in the ground truth, which are underrepresented in the predictions.

5.3.4 Runtime Performance

The average prediction time of our model for a single gaze position is $\approx 500ms$. This is currently a limitation for real-time applications, such as foveated rendering in virtual reality. Ideally, the model should operate at speeds comparable to or faster than traditional eye trackers ($\approx 16 - 33ms$ at frame rate of 30-60Hz), enabling it to either replace or supplement them to address latency issues in gaze-based rendering.

The primary factor contributing to the slower runtime is the use of ConvLSTM layers, which, while effective for capturing temporal dependencies, are computationally intensive. In addition, the model's overall complexity, due to the presence of multiple layers, further worsens the inference time. While these design choices were necessary to achieve the desired accuracy, they have significantly impacted the model's runtime performance. This performance bottleneck suggests that further optimization is needed, potentially exploring alternative architectures or reducing the model's complexity to meet the real-time requirements of latency-sensitive applications. However, this may require trade-offs between model accuracy and processing speed, which will need to be carefully considered.

5.3.5 Visualisation

Finally, we visualised the results to obtain visual feedback. In the following figure, we present several different cases. The main point we emphasize is that the predictions of our model are almost always within the foveal circle around the ground truth gaze point. Additionally, it is easy to observe that the predictions generally follow the pattern of the ground truth, even without achieving perfect accuracy.

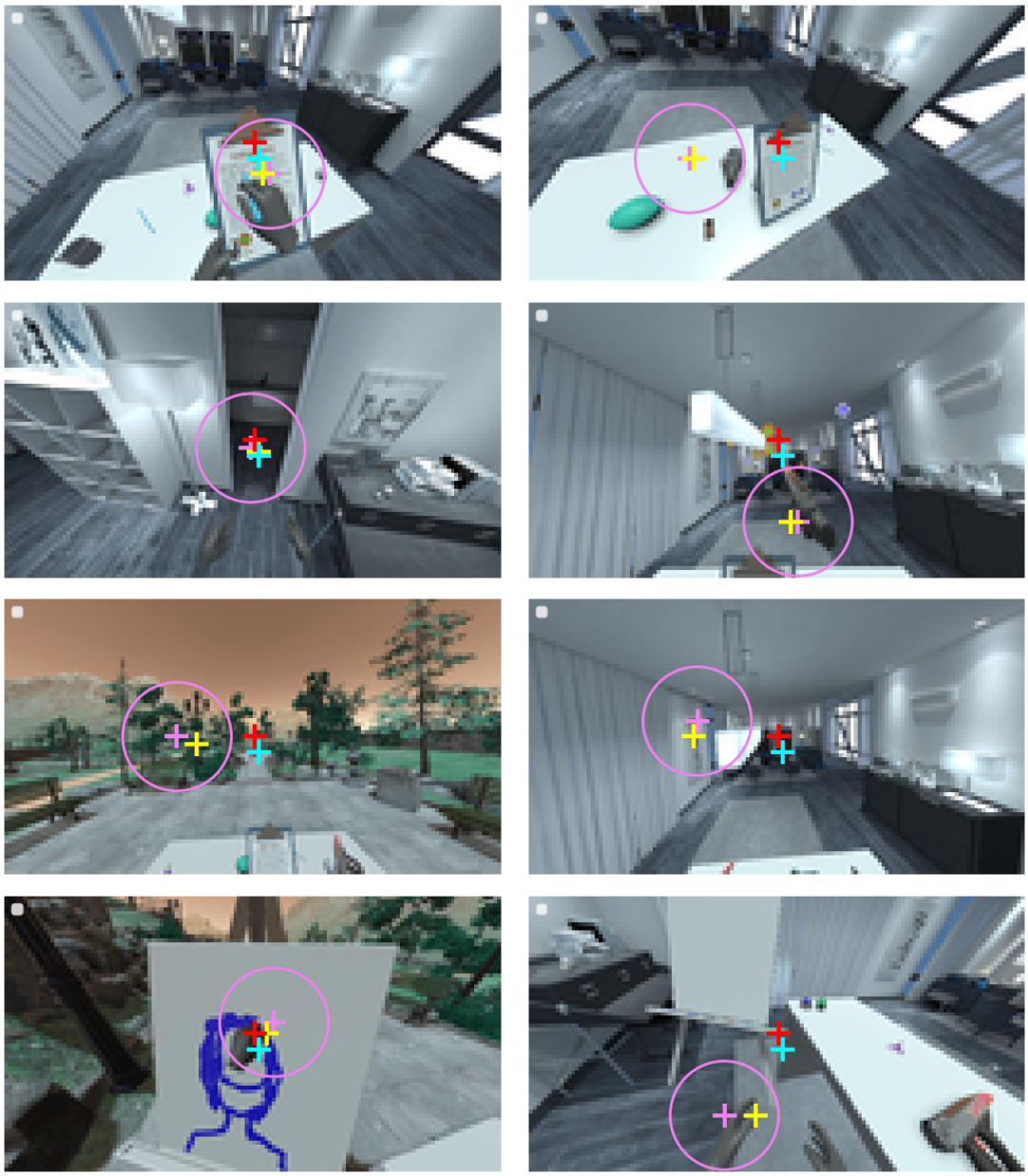


Figure 5.4: Visualisation of our results. The purple (+) cross denotes the ground truth gaze position, with the purple circle illustrating the foveal region with radius 15°. The yellow (+) cross represents the prediction of our model, the red (+) cross shows the center baseline and the blue (+) the mean baseline.

Chapter 6

Conclusions and Future Work

In this chapter, we will summarize the conclusions drawn from the implementation of this thesis. We will highlight the strengths of our approach and the achievements made, as well as discuss the limitations of the system. Finally, we will outline potential directions for future work to further improve the results.

6.1 Conclusions

In this thesis, a gaze prediction model for task-oriented VR environments based on deep neural networks was developed. The model consists of three modules: the *image sequence module*, which learns the temporal motion features among consecutive frames; the *gaze sequence module*, which captures the temporal patterns of user gaze; and the *fusion module*, which combines the outcomes of the first two modules to make a single gaze prediction. Our research highlighted key limitations in existing gaze prediction models, including the need for task-specific focus in VR, generalized adaptability across tasks, simplified input requirements for integration, and lightweight architectures capable of rapid predictions. This led us to explore the relationship between task-specific environments and temporal continuity as the foundation for a more accurate and flexible model.

To train the model, we used the OpenNEEDS dataset, a newly published large-scale, open-source dataset containing data from 44 users who freely explored virtual environments and performed various tasks.

The model was evaluated based on its prediction error (measured in degrees), its recall rate to assess potential use in foveated rendering pipelines, and its runtime performance.

Two baselines (center and mean) were defined for comparison. Using the angular distance between the predicted gaze position and the ground truth, we compared the model to the baselines in terms of error and recall rate. The results are promising. The model demonstrates a very low median error of approximately 4 degrees, with a small IQR, indicating robustness, accuracy, and consistency. Furthermore, the model achieved a recall rate of 99.87%, suggesting its practical applicability in assisting or replacing an eye tracker.

However, the average runtime performance remains a significant limitation of the system. To achieve the desired accuracy, the model's architecture is complex and computationally intensive. Considerable optimization is required to make the model more lightweight and faster, even if it necessitates trading off some accuracy for processing speed.

Overall, our results show that temporal continuity is a solid approach to addressing the problem of gaze prediction. The model not only achieves excellent accuracy and consistency but does so with a dataset consisting of a wide variety of tasks, showcasing its adaptability. However, further improvements are necessary to optimize runtime performance, enabling its use in real-time rendering pipelines.

6.2 Future Work

The main limitation of this thesis is the runtime performance of the developed model. There are several approaches that could be explored to address this issue.

First, reducing the model's complexity while retaining the core logic of the architecture is a promising avenue. The image sequence module is the most computationally intensive component, as it consists of multiple ConvLSTM2D layers, which are particularly resource-heavy. Reducing the number of layers or simplifying the ConvLSTM2D layers—for instance, by decreasing the number of filters (currently set at 16)—could help. Improved pooling strategies, such as experimenting with different pooling sizes or switching to average pooling, might also contribute to efficiency gains. The gaze sequence module, while less demanding, could also benefit from simplification, such as reducing the number of units in the LSTM layers. Additionally, using shorter input sequences could significantly enhance runtime performance.

If the model is optimized to be fast enough, it should be integrated and tested in an actual foveated rendering pipeline. While most state-of-the-art works do not explore this integration due to suboptimal results, conducting such experimentation is crucial. It will allow for identifying the challenges that arise in real-world applications and provide insights into possible solutions. Testing within the pipeline would also help in refining the model for practical, real-time usage.

Beyond runtime improvements, there is still room to improve the model's accuracy. Fine-tuning training parameters through experimentation may further boost the performance. Training on diverse datasets could also improve the model's generalizability across various environments and tasks. Exploring more sophisticated fusion techniques, such as attention mechanisms or weighted concatenation, might also optimize the fusion module's performance.

Alternatively, entirely different approaches could be considered. Convolutional layers, while effective for processing image inputs, are computationally expensive. A simpler architecture not based on frames might be viable. For instance, incorporating alternative inputs like head and hand movement—a concept gaining traction in recent literature—could offer a lightweight and potentially more accurate solution. This data is less computationally intensive, which could make the model faster while maintaining or even improving its accuracy. Another fascinating and underexplored area is the role of sound in gaze behaviour and its implications for gaze prediction. Investigating this would require a significant amount of research to understand how sound influences gaze and to develop datasets that integrate audio information.

In conclusion, gaze prediction is a field with great potential for further research. By solving current challenges and trying out new ideas, we can create models that are more accurate, faster, and flexible, with many practical uses in different areas.

References

[1] Norman, J.: The sensorama: One of the first functioning efforts in virtual reality. https://www.historyofinformation.com/detail.php?id=2785 ix, 8

[2] Azif Ismail, P.J.S.P.: Virtual reality: Introduction. https://www.dsource.in/course/virtual-reality-introduction ix, 8

[3] Sorene, P.: Jaron lanier's eyephone: Head and glove virtual reality in the 1980s. https://flashbak.com/jaron-laniers-eyephone-head-and-glove-virtual-reality-in-the-1980s-26180/ ix, 9

[4] Wiki: Oculus rift. https://el.wikipedia.org/wiki/Oculus_Rift ix, 9

[5] Valve: Valve index. https://www.valvesoftware.com/en/index ix, 10

[6] Meta: Meta quest 3. https://www.meta.com/quest/quest-3/ ix, 11

[7] HTC: Htc vive pro eye specs. https://www.vive.com/sea/product/vive-pro-eye/specs/ ix, 11

[8] Guenter, B., Finch, M., Drucker, S., Tan, D., Snyder, J.: Foveated 3d graphics. ACM Transactions on Graphics (TOG) **31**(6) (2012) 1–10 ix, 13

[9] Vedantu: Eye structure. https://www.vedantu.com/biology/structure-of-eye ix, 14

[10] Environmental Protection Department, Hong Kong: Eia report: Appendix 11.1 (2017) Accessed: September 6, 2024. ix, 17

[11] Hu, Z., Li, S., Zhang, C., Yi, K., Wang, G., Manocha, D.: Dgaze: Cnn-based gaze prediction in dynamic scenes. IEEE Transactions on Visualization and Computer Graphics **26**(5) (2020) 1902–1911 ix, 20, 21, 29, 65, 69, 79

[12] Emery, K.J., Zannoli, M., Warren, J., Xiao, L., Talathi, S.S.: Openneeds: A dataset of gaze, head, hand, and scene signals during exploration in open-ended vr environments. In: ACM Symposium on Eye Tracking Research and Applications. ETRA '21 Short Papers, New York, NY, USA, Association for Computing Machinery (2021) ix, x, 20, 50, 51

[13] Haji-Abolhassani, A., Clark, J.: A computational model for task inference in visual search. Journal of vision **13** (02 2013) x, 22

[14] Labs, P.: Pupil labs core. `https://pupil-labs.com/products/core` x, 23

[15] Nguyen, T.H., Nguyen, T.N., Ngo, B.V.: A vgg-19 model with transfer learning and image segmentation for classification of tomato leaf disease. AgriEngineering **4** (10 2022) 871–887 x, 26

[16] Huang, X., Shen, C., Boix, X., Zhao, Q.: Salicon: Reducing the semantic gap in saliency prediction by adapting deep neural networks. In: Proceedings of the IEEE International Conference on Computer Vision (ICCV). (December 2015) x, 26, 27

[17] Cornia, M., Baraldi, L., Serra, G., Cucchiara, R.: Predicting human eye fixations via an lstm-based saliency attentive model. IEEE Transactions on Image Processing **27**(10) (October 2018) 5142–5154 x, 20, 27, 28, 30

[18] Sitzmann, V., Serrano, A., Pavel, A., Agrawala, M., Gutierrez, D., Masia, B., Wetzstein, G.: How do people explore virtual environments? IEEE Transactions on Visualization and Computer Graphics (2017) x, 19, 20, 21, 28, 29

[19] Dertat, A.: Applied deep learning - part 1: Artificial neural networks. `https://towardsdatascience.com/applied-deep-learning-part-1-artificial-neural-networks-d7834f67a4f6` x, 34

[20] Mongaras, G.: How do neural networks work? `https://gmongaras.medium.com/how-do-neural-networks-work-bfdd3ca6c23a` x, 34

[21] Haque, K.N.: What is convolutional neural network — cnn (deep learning). https://www.linkedin.com/pulse/what-convolutional-neural-network-cnn-deep-learning-nafiz-shahriar x, 38

[22] Pokhrel, S.: Beginners guide to convolutional neural networks. https://towardsdatascience.com/beginners-guide-to-understanding-convolutional-neural-networks-ae9ed58bb17d x, 39

[23] Yehia, N.: Beginners guide to convolutional neural networks. https://mlarchive.com/deep-learning/understanding-long-short-term-memory-networks/ x, 41

[24] Shi, X., Chen, Z., Wang, H., Yeung, D.Y., Wong, W.K., Woo, W.c.: Convolutional lstm network: A machine learning approach for precipitation nowcasting. Advances in neural information processing systems **28** (2015) x, 43, 44

[25] Heilig, M.: Sensorama. https://en.wikipedia.org/wiki/Sensorama (1962) 7

[26] Society, V.R.: History of vr. https://www.vrs.org.uk/virtual-reality/history.html 8

[27] Research, V.: Vpl research. https://www.vrs.org.uk/virtual-reality-profiles/vpl-research.html 9

[28] Speech, V.: History of vr. https://virtualspeech.com/blog/history-of-vr 9

[29] Wang, L., Shi, X., Liu, Y.: Foveated rendering: A state-of-the-art survey. Computational Visual Media **9** (2023) 195–228 12, 13

[30] Patney, A., Salvi, M., Kim, J., Kaplanyan, A., Wyman, C., Benty, N., Luebke, D., Lefohn, A.: Towards foveated rendering for gaze-tracked virtual reality. ACM Trans. Graph. **35**(6) (December 2016) 12, 79

[31] Riordan-Eva, P., Whitcher, J.P.: Vaughan & asbury's general ophthalmology. (2011) 15

[32] : Adler's physiology of the eye: Clinical application. (2011) 15

[33] Kolb, H., Fernandez, E., Nelson, R.: Webvision: The organization of the retina and visual system. (2005) Available online: https://webvision.med.utah.edu/. 15

[34] Palmer, S.E.: Vision Science: Photons to Phenomenology. MIT Press, Cambridge, MA, USA (1999) 15

[35] Howard, I.P., Rogers, B.J.: Perceiving in Depth, Volume 1: Basic Mechanisms. Oxford University Press, New York, NY, USA (2012) 16

[36] Blake, R., Wilson, H.R.: Visual Perception: Physiology, Psychology, and Ecology. Psychology Press, New York, NY, USA (2011) 16

[37] Regan, D.: Human Perception of Objects. Taylor & Francis Group, London, UK (2000) 16

[38] Goldstein, E.B.: Sensation and Perception. 9th edn. Wadsworth Cengage Learning, Belmont, CA, USA (2014) 16

[39] Rayner, K.: Eye movements and attention in reading, scene perception, and visual search. Quarterly Journal of Experimental Psychology **62**(8) (2009) 1457–1506 17, 18

[40] Holmqvist, K., Nyström, M., Andersson, R., Dewhurst, R., Jarodzka, H., van de Weijer, J.: Eye Tracking: A Comprehensive Guide to Methods and Measures. Oxford University Press (2011) 17, 18

[41] Salvucci, D.D., Goldberg, J.H.: Identifying fixations and saccades in eye-tracking protocols. Proceedings of the Eye Tracking Research and Applications Symposium (2000) 71–78 17

[42] Alexander C. Schütz, David I. Braun, K.R.G.: Eye movements and perception: A selective review. Journal of Vision **11**(5) (2011) 9–9 17, 18

[43] Macknik, S.L., Martinez-Conde, S.: The Neurobiology of Saccades and Fixations. Oxford University Press (2008) Chapter 4. 18

[44] Roy S. Hessels, Diederick C. Niehorster, I.T.C.H., der Stigchel, H.C.A.C.V., van der Geest, J.J.F.: Is the eye-movement field confused about fixations and saccades? a survey among 124 researchers. Royal Society Open Science **5**(8) (2018) 180502 18

[45] Itti, L., Koch, C.: Computational modelling of visual attention. Nature Reviews Neuroscience **2**(3) (2001) 194–203 19, 25

[46] Bruce, N., Tsotsos, J.: Saliency, attention, and visual search: An information theoretic approach. Journal of vision **9** (03 2009) 5.1–24 19

[47] Zhang, L., Lin, W.: Selective Visual Attention: Computational Models and Applications. 1st edn. Wiley-IEEE Press (2013) 19

[48] Tatler, B.W.: The central fixation bias in scene viewing: Selecting an optimal viewing position independently of motor biases and image feature distributions. Journal of Vision **7**(14) (11 2007) 4–4 19

[49] Tseng, P.H., Carmi, R., Cameron, I.G.M., Munoz, D.P., Itti, L.: Quantifying center bias of observers in free viewing of dynamic natural scenes. Journal of Vision **9**(7) (07 2009) 4–4 19

[50] Hu, Z., Zhang, C., Li, S., Wang, G., Manocha, D.: Sgaze: A data-driven eye-head coordination model for realtime gaze prediction. IEEE Transactions on Visualization and Computer Graphics **25**(5) (2019) 2002–2010 21, 69

[51] Freedman, E.G.: Coordination of the eyes and head during visual orienting. Experimental Brain Research **190**(4) (2008) 369–387 21

[52] Sundstedt, V., Stavrakis, E., Wimmer, M., Reinhard, E.: A psychophysical study of fixation behavior in a computer game. In: Proceedings of the 5th Symposium on Applied Perception in Graphics and Visualization. APGV '08, New York, NY, USA, Association for Computing Machinery (2008) 43–50 21

[53] Logan, G.D., Cowan, W.B., Davis, K.A.: On the ability to inhibit simple and choice reaction time responses: a model and a method. Journal of experimental psychology: human perception and performance **10**(2) (1984) 276 21

[54] Hu, Z., Li, S., Gai, M.: Temporal continuity of visual attention for future gaze prediction in immersive virtual reality. Virtual Reality & Intelligent Hardware **2**(2) (2020) 142–152 22, 31

[55] Carter, B.T., Luke, S.G.: Best practices in eye tracking research. International Journal of Psychophysiology **155** (2020) 49–62 23

[56] Tobii: Tobii pro fusion. `https://www.tobii.com/products/eye-trackers/screen-based/tobii-pro-fusion` 23

[57] Krasovskaya, S., MacInnes, W.J.: Salience models: A computational cognitive neuroscience review. Vision **3**(4) (2019) 56 25

[58] Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014) 25, 39

[59] Kümmerer, M., Wallis, T.S., Bethge, M.: Deepgaze ii: Reading fixations from deep features trained on object recognition. arXiv preprint arXiv:1610.01563 (2016) 26

[60] He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). (June 2016) 27

[61] Cornia, M., Baraldi, L., Serra, G., Cucchiara, R.: Multi-level net: A visual saliency prediction model. In: Computer Vision–ECCV 2016 Workshops: Amsterdam, The Netherlands, October 8-10 and 15-16, 2016, Proceedings, Part II 14, Springer (2016) 302–315 29

[62] Pan, J., Ferrer, C.C., McGuinness, K., O'Connor, N.E., Torres, J., Sayrol, E., Giro-i Nieto, X.: Salgan: Visual saliency prediction with generative adversarial networks. arXiv preprint arXiv:1701.01081 (2017) 29

[63] Koulieris, G.A., Drettakis, G., Cunningham, D., Mania, K.: Gaze prediction using machine learning for dynamic stereo manipulation in games. In: 2016 IEEE virtual reality (VR), IEEE (2016) 113–120 29

[64] Hu, Z., Bulling, A., Li, S., Wang, G.: Fixationnet: Forecasting eye fixations in task-oriented virtual environments. IEEE Transactions on Visualization and Computer Graphics **27**(5) (2021) 2681–2690 31, 69

[65] LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. nature **521**(7553) (2015) 436–444 33

[66] Heaton, J.: Ian goodfellow, yoshua bengio, and aaron courville: Deep learning: The mit press, 2016, 800 pp, isbn: 0262035618. Genetic programming and evolvable machines **19**(1) (2018) 305–307 33, 34, 35

[67] Bishop, C.M.: Neural networks for pattern recognition. Oxford university press (1995) 33, 36

[68] Baheti, P.: Activation functions in neural networks [12 types and use cases]. `https://www.v7labs.com/blog/neural-networks-activation-functions` 35

[69] Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning representations by back-propagating errors. nature **323**(6088) (1986) 533–536 36

[70] Willmott, C.J., Matsuura, K.: Advantages of the mean absolute error (mae) over the root mean square error (rmse) in assessing average model performance. Climate research **30**(1) (2005) 79–82 36

[71] Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization (2017) 37

[72] LeCun, Y., Bengio, Y., et al.: Convolutional networks for images, speech, and time series. The handbook of brain theory and neural networks **3361**(10) (1995) 1995 37

[73] Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems **25** (2012) 38, 40

[74] LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE **86**(11) (1998) 2278–2324 38

[75] Zeiler, M.D., Fergus, R.: Visualizing and understanding convolutional networks. In: Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I 13, Springer (2014) 818–833 38, 39

[76] Scherer, D., Müller, A., Behnke, S.: Evaluation of pooling operations in convolutional architectures for object recognition. In: International conference on artificial neural networks, Springer (2010) 92–101 40

[77] Hochreiter, S.: Long short-term memory. Neural Computation MIT-Press (1997) 41

[78] Apache: Apache parquet file. https://parquet.apache.org/#td-block-1 46

[79] Folk, M., Heber, G., Koziol, Q., Pourmal, E., Robinson, D.: An overview of the hdf5 technology suite and its applications. In: Proceedings of the EDBT/ICDT 2011 workshop on array databases. (2011) 36–47 46

[80] Google: Google colaboratory. https://colab.google/ 47

[81] TensorFlow: Keras: The high-level api for tensorflow. https://www.tensorflow.org/guide/keras 47

[82] OpenCV: Opencv (open source computer vision library). https://opencv.org/ 47

[83] ETRA '21 Short Papers: ACM Symposium on Eye Tracking Research and Applications, New York, NY, USA, Association for Computing Machinery (2021) 50