Assistive App Based on Eye Tracking
基于眼动追踪的辅助应用

a thesis

submitted to School of Mathematics & Physics

of Xi'an Jiaotong-Liverpool University

in partial fulfilment for the award of the degree of

BSc Applied Mathematics

By

Yu Qiu 2036484

Supervisor: Dr. Arodh Lal Karn

May 7, 2024

Abstract

The objective of this thesis is to explore the feasibility and effectiveness of implementing a real-time eye-tracking system using the YOLOv5 object detection model, optimized for mobile platforms through TensorFlow Lite. This research addresses the challenge of deploying high-performance deep learning models on resource-constrained devices, such as smartphones, by utilizing model compression techniques within TFLite to reduce computational demand without substantially sacrificing accuracy.

The implementation of YOLOv5, adapted for eye-tracking, demonstrates the model's versatility and robustness in detecting eye positions under various environmental conditions. The system leverages TensorFlow Lite's capabilities for model quantization and optimization to ensure that the application runs efficiently in real-time on Android devices. The results indicate that the optimized YOLOv5 model achieves satisfactory accuracy and latency on standard mobile hardware, making it suitable for a wide range of applications from assistive technologies to user behavior analysis.

This study not only contributes a practical solution to mobile-based eye-tracking but also provides a foundation for future research in enhancing the accessibility and usability of eye-tracking technologies through advanced machine learning techniques. Further explorations into adaptive quantization and dynamic model adjustments could offer improvements in system performance and energy efficiency, broadening the potential use cases for mobile-based eye-tracking solutions.

本论文的目标是探讨使用 YOLOv5 目标检测模型实现基于移动平台的实时眼动追踪系统的可行性和有效性，该系统通过 TensorFlow Lite 进行优化。本研究解决了在资源受限的设备上部署高性能深度学习模型的挑战，如智能手机，通过在 TFLite 中使用模型压缩技术来减少计算需求，而不会大幅牺牲准确性。

YOLOv5 模型经过适应眼动追踪的实现，展示了该模型在各种环境条件下检测眼部位置的多功能性和鲁棒性。系统利用 TensorFlow Lite 的模型量化和优化功能，确保应用程序能够在 Android 设备上高效地实时运行。结果表明，优化后的 YOLOv5 模型在标准移动硬件上实现了令人满意的准确性和响应速度，适用于从辅助技术到用户行为分析的广泛应用。

这项研究不仅为基于移动设备的眼动追踪提供了一种实际的解决方案，还为通过先进的机器学习技术增强眼动追踪技术的可接近性和可用性提供了研究基础。进一步探索适应性量化和动态模型调整可能会改善系统性能和能效，拓宽基于移动设备的眼动追踪解决方案的潜在应用场景。

KEY WORDS: YOLOv5, TensorFlow Lite, Eye-tracking, Real-time object detection, Mobile vision applications, Deep learning optimization, Android development

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Dr. Arodh Lal Karn, for his substantial support and invaluable advice on my research project. Dr. Karn provided not only expert guidance but also encouragement and support in my academic pursuits and personal growth, enabling me to confidently tackle challenges and overcome obstacles.

Additionally, I am deeply thankful to the members of the open-source community. Their selfless contributions have greatly enriched the global technology resource pool, providing essential tools and support for this research. The developers behind these open-source projects demonstrate the importance of collaboration and knowledge sharing, which were crucial for the completion of this project.

I also wish to thank all colleagues and friends who participated in testing and provided feedback. Their insights helped refine my methodology and experimental design, ensuring the practicality and relevance of the research.

Lastly, my family deserves special recognition for their understanding and support of my academic endeavors. Their encouragement is the driving force behind my continuous pursuit of scholarly research.

Once again, thank you to everyone who has directly or indirectly assisted and supported me. This work could not have been accomplished without your help.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The evolution of eye-tracking technology can be segmented into three distinct periods:

Pre-2000: Originating in the 19th century, research on human gaze points was mainly confined to academia, exploring physiological, psychological, and ophthalmological aspects to understand eye function and information processing, both consciously and subconsciously.

2000-2020: As the internet economy grew, the technology became more compact and lightweight, finding applications in web usability, advertising, and marketing.

2020-Present: Applications have diversified, with consumer-grade devices in XR and AE gaining market presence. Despite many solutions, issues persist with cost, convenience, and hardware dependency.

Today, despite the numerous existing solutions, many of them come with one or more issues: they are either high-cost, cumbersome to use, or require specific hardware facilities.

We aim to develop a convenient and practical lightweight eye-tracking Android app. It doesn't require high accuracy or complex functionality but aims to have a low entry barrier for ease of use.

This app needs to meet the following requirements:
1. The eye-tracking technology should be suitable for use on mobile devices, addressing the two-dimensional eye-tracking problem.
2. Given the limited performance and battery life of mobile platforms, the application must have the capability to keep performance utilization as low as possible, especially since excessive resource consumption can hinder normal usage.
3. It should provide a certain level of convenience for users.

The intersection of mobile computing and real-time object detection has seen remarkable progress with the advent of efficient deep learning models like YOLOv5 and advancements in model op-

timization techniques such as TensorFlow Lite (TFLite). This thesis explores the integration of YOLOv5 within the Android ecosystem, optimized using TFLite to harness the benefits of low-latency and high-efficiency object detection on resource-constrained mobile devices. The study leverages these technologies to develop a robust eye-tracking application, demonstrating the feasibility and practicality of deploying advanced neural networks in everyday applications to enhance interactive experiences.

Chapter 2

Literature Review

2.1 Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference(Gholami et al. (2021))

The paper details several methods for quantizing neural networks, including uniform and non-uniform quantization, symmetric and asymmetric quantization, and techniques such as scalar and vector quantization. These methods vary in how they convert continuous real numbers into discrete numeric values, impacting the model's memory footprint and computational efficiency. Quantizing neural networks can reduce the memory footprint and computational resources required for inference. For instance, using four-bit representations can potentially decrease the memory and computation needs by up to 16 times compared to floating-point representations. The document discusses different granularities of applying quantization, such as per-layer, per-channel, or per-group quantization, which can optimize the balance between compression and performance across various parts of the network. While quantization can significantly reduce computational requirements, it also poses challenges for maintaining model accuracy. Techniques like Quantization-Aware Training (QAT) and mixed-precision quantization strategies are discussed as methods to mitigate accuracy loss.

2.2 A Survey on Deep Neural Network Compression: Challenges, Overview, and Solutions(Mishra et al. (2020))

The paper "A Survey on Deep Neural Network Compression" provides an extensive review of various techniques aimed at optimizing neural networks for deployment on resource-constrained devices such as IoT platforms. It categorizes compression methods into five key areas: include knowledge distillation, network pruning, bits precision, sparse representation, and miscellaneous techniques—each addressing different strategies for reducing model size and computational demand while striving to maintain accuracy. The discussion elaborates on the challenges of balancing compression intensity with performance retention and underscores the importance of these compression techniques in facilitating the use of advanced machine learning in environments

with limited computational resources. The review serves as a critical resource for researchers and practitioners focused on enhancing the efficiency and accessibility of machine learning technologies in resource-limited settings.

2.3 A Survey on Eye-Gaze Tracking Techniques(Chennamma and Yuan (2013)

The interrelation between the development of eye-tracking technology and the advancement in fields such as computer vision and human-computer interaction is pivotal. This survey delineates the chronological advancements of eye-tracking methodologies. Initially, eye-tracking necessitated intrusive methods that involved physical contact. Modern developments have ushered in non-intrusive, video-based tracking systems, vastly enhancing user convenience and broadening the application spectrum. Furthermore, there has been a significant evolution from rudimentary laboratory configurations to intricate systems conducive to dynamic user engagement across diverse settings. The discourse accentuates the necessity for ongoing research and the establishment of standards to ameliorate the precision, durability, and economic viability of eye-tracking systems, thus widening their accessibility for commercial exploitation and scholarly inquiry.

2.4 End-to-End Object Detection with Transformers(Carion et al. (2020))

The paper introduces a novel encoder-decoder transformer architecture tailored for object detection. Transformers are used due to their self-attention mechanisms, in which model all pairwise interactions between elements in just one sequence, making them particularly suitable for tasks like object detection where modeling of global dependencies is crucial.What's more, DETR demonstrates competitive performance with Faster R-CNN on the challenging COCO dataset. It shows very significantly better performance on most of large objects, which the authors attribute to the non-local computations of transformers.

2.5 YOLOv5(You Only Look Once)(Ultralytics (2020))

YOLOv5, developed by Ultralytics, is a top-tier object detection model that belongs to the YOLO (You Only Look Once) family. This model excels in identifying and positioning objects with a single image analysis, significantly boosting processing speed which is crucial for real-time applications like traffic monitoring, autonomous driving, and security systems.

Available in various versions such as YOLOv5s, YOLOv5m, YOLOv5l, and YOLOv5x, each model offers a different trade-off between speed and accuracy, catering to diverse requirements and hardware capabilities. For example, the smaller models are faster and fit for less powerful devices, while the larger ones are more accurate, suitable for high-precision tasks.

Technically, YOLOv5 employs anchor boxes for initial bounding box predictions and refines these to match actual object sizes and locations. Its composite loss function—which includes

class, object, and box losses—optimizes performance across multiple environments, ensuring dependable operation under various conditions.

2.6 Eye Tracking for Everyone(Krafka et al. (2016))

The article establishes a database for eye-tracking with a focus on mobile devices and provides training guidelines for the corresponding CNN neural network model.
It also highlights a series of benefits associated with using mobile devices for eye-tracking:
(1) Widespread Usage: It is estimated that over one-third of the world's population was using smartphones in 2019, far surpassing the number of desktop or laptop users.
(2) High Adoption of Technological Upgrades: A significant portion of the population possesses the latest hardware, allowing for real-time utilization of computationally expensive methods such as Convolutional Neural Networks (CNNs).
(3) Rapid Advancements in Camera Technology: The extensive use of cameras on mobile devices has led to rapid development and deployment of camera technologies.
(4) Fixed Camera Position Relative to the Screen: The fixed position of the camera in relation to the screen reduces the number of unknown parameters.

2.7 Eye Tracking in User Experience Design(Bergstrom and Schall (2014))

Classification of Eye-Tracking Interactive Applications:
There are four types of eye-tracking interactive applications: active, passive, expressive, and diagnostic.
1. Active Interaction: It serves as an input method based on user intent, such as using eye gaze to control electronic devices.
2. Passive Interaction: This technology optimizes user experience by tracking eye gaze positions in real-time.
3. Expressive Interaction: It uses eye-tracking to drive digital avatars, reducing the uncanny valley effect.
4. Diagnostic Applications: Eye-tracking technology is used to diagnose or analyze individuals or groups for scientific research or optimization of business activities. For example, analyzing the focal points of the audience in the advertising industry to enhance ad effectiveness.

Eye Tracking on Mobile Devices:
On different sizes of mobile devices, users' browsing experiences on the same website vary. According to the content of the paper(Seix et al. (2012)),conducted research on whether the screen size of mobile devices affects user eye movements. The results showed that smaller screens made reading more difficult and, as a result, prolonged gaze duration.

Chapter 3

Methodology

3.1 Eye-Tracking Principles

To understand the principles of eye tracking, it is essential to begin with a physiological analysis of the types and mechanisms of eye movements:

The human eye can move within six degrees of freedom: three translational movements within the eye socket and three rotational movements.

- Horizontal rotation ranges from 45° to 55°. - Vertical rotation ranges from 47° to 55° upward and from 28° to 35° downward. - With increasing age, the range of rotational motion may decrease (Lee et al. (2019)).

Types and Classification of Eye Movements: Eye movements can be categorized into several types, including saccades, smooth pursuit, vergence, vestibular, and fixations (essentially physiological nystagmus). Among these, only fixations, smooth pursuit, and saccades involve conscious attention shifts. Therefore, processing is mainly focused on these three types of eye movements:

1. Fixations. 2. Smooth Pursuit. 3. Saccades.

Saccades: Saccades are rapid eye movements used to shift the central point of vision to a new location within the visual environment. They are characterized by their high speed, and once initiated, they do not change the visual target during the movement. Therefore, saccades can be relatively easily predicted.

A saccade signal can be represented using a differential filter.

The process of series expansion is a fundamental technique in mathematical analysis. It allows the approximation of complex functions by a series of simpler terms. The expansion of a function x can be expressed as a sum of terms:

$$x_t = g_0 s_t + g_1 s_{t-1} + \ldots = \sum_{k=0}^{\infty} g_k s_{t-k} \tag{3.1}$$

Where s_t denotes the signal at time t, and g_k represents the weight of the k-th term in the expansion. This series form, particularly when considering the Haar wavelet transformation, introduces an elegant method to dissect signals into hierarchical components.

Further, we define the transformation for a signal X and its Haar wavelet series S as follows:

$$
\begin{aligned}
x_t &= g_0 s_t + g_1 s_{t-1}, \\
x_t &= (1)s_t + (-1)s_{t-1}, \\
x_t &= (1)s_t + (-1)s_{t-1}, \\
x_t &= (1-z)s_t, \\
X(z) &= (1-z)S(z), \\
X(z) &= \frac{S(z)}{1-z}.
\end{aligned}
$$

The Haar wavelet transform provides a versatile tool for signal analysis, offering insights into the inherent structure of the data.

Smooth Pursuit: During smooth pursuit, the eyes visually track a target, matching the movement of the target based on its motion range. It involves a pursuit behavior where the eyes follow the motion of the target.
Fixations: Fixation on a stationary target does not mean that the eyes are entirely motionless. Fixations are characterized by small eye movements and are eye movements that vary within a range of 1-2 degrees of arc. It is not the case that the eyes remain completely still as commonly perceived. In fact, if an image is fixed on the retina, vision fades within about one second, and the scene becomes blank (Leigh and Zee (1991)).
In human-computer interaction, these three behaviors can be categorized into two behavioral metrics: fixations and saccades. Fixations refer to when the eyes remain stationary in a fixed area for a period, typically 200-300 milliseconds. However, fixations do not mean the eyes are completely still, and they may involve slight eye movements (tremors, drifts, and microsaccades) during this process. Saccades, on the other hand, involve the eyes jumping between different fixation points, with the jump amplitude ranging from 1° to 45°. Generally, if the angle of the jump exceeds 30°, it is often accompanied by head movement to increase efficiency.

Saccade Detection

Velocity-based Saccade Detection(Anliker (1976)): Within a sampling window, the velocity of the signal is calculated and compared to a velocity threshold. If it is below the given threshold, it is considered a fixation signal; otherwise, it is a saccade signal.

Eye-tracking analysis

Once we have collected the eye-tracking data from observers, how should we utilize it? The purpose of eye-tracking analysis is to gain insights into the observers' attention behavior.
The eye-tracking data we collect is a form of signal. We need to interpret it as specific eye movements, such as saccades, smooth pursuit, and fixations (essentially physiological nystagmus). Describing these specific types of eye movements is crucial for understanding the observers' attention behavior and making appropriate adjustments.

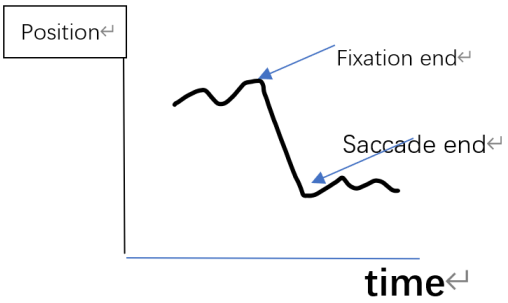

Figure 3.1: Fixation and Saccade

There are two automatic methods commonly used for eye-tracking analysis:
1. **Averaging-Based Method**: This approach segments the eye-tracking data over a period and calculates the average within those segments. If the differences between segments are small or non-existent, they are marked as potential fixation points. If the signal persists for a duration exceeding a predefined threshold, it is considered a fixation point.
2. **Differential-Based Method**(Yarbus (1967)): In this method, eye-tracking signal positions are recorded at fixed time intervals, and the velocity of eye movement is estimated by calculating the difference in position divided by time. In fixation states, there is minimal change in position, while saccades and smooth pursuits result in rapid changes in the eye-tracking signal's position, thus helping determine shifts in the observer's attention.

How to process eye-tracking signals

Theorem 3.1 (To remove noise from eye-tracking signals.) *Due to the instability of the eyes, such as blinking, some signal processing should be applied to the acquired eye-tracking data to eliminate noise.*
Noise Reduction Method: A given small rectangle is used to perform detection on each point. Data that falls outside the range of variation of the rectangle will be ignored.

The development of eye-tracking technology has been instrumental in a myriad of applications, from psychological insights to advanced human-computer interaction. It hinges on the precise capture of gaze data, which is critical for both academic research and commercial utility. However, the application of this technology encounters practical limitations such as high costs and operational complexity, necessitating the exploration of more accessible approaches.
In the domain of gaze data analysis, the recent advancements have led to the formulation of an aggregated precision computation model. This model integrates the gaze data over a rolling window of the past n instances, ensuring a harmonious balance between real-time responsiveness and historical accuracy. The equation below encapsulates the model:

$$PC_{new} = \frac{1}{n}\sum_{i=0}^{n-1} PC_{m-i} \tag{3.2}$$

Here, PC_{new} signifies the updated precision calculation, and PC_{m-i} denotes the precision score at each historical instance.

Moreover, the utility of glints in refining gaze estimation accuracy is acknowledged. Glints are reflections that assist in stabilizing the eye-tracking process, offering a secondary data point for enhancing precision. The performance of the system is thus dual-faceted, evaluating both the accuracy and precision to provide a comprehensive measure of the eye-tracking efficacy.

Corollary 3.1.1 *Differential-Based Observational Method: This method involves recording the position of eye-tracking signals at fixed time intervals and estimating the velocity of eye movement by calculating the difference in position divided by time. In fixation states, there is minimal change in position, while saccades and smooth pursuits result in rapid changes in the position of eye-tracking signals. This helps determine shifts in the observer's attention.*

Lemma 3.2 *Based on Velocity-based Saccade Detection:*
Within a sampling window, calculate the velocity of the signal and compare it to a velocity threshold. If it is below the given threshold, it is considered a fixation signal; otherwise, it is a saccade signal.
Firstly, according to observations in the paper (Anliker, 1976), the velocity of saccades appears as a bell-shaped curve, essentially symmetric. Therefore, once a velocity peak is detected, the position of the next fixation point can be estimated based on previous records (see saccadic jumps in eye-tracking principles).

$$v = \frac{\sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2}}{dt} \tag{3.3}$$

How should this program operate?

The system will consist of four components:
1. Calibration Program: This program performs initial data collection to enhance accuracy. It collects basic information about the mobile device, such as camera position and hardware level, to optimize accuracy. It may also attempt to gather user-specific information.
2. Recognition Program: This program is responsible for identifying and collecting data. It utilizes Convolutional Neural Networks (CNNs) for prediction and can predict head orientation and gaze direction. It may utilize established data and models available in the market, such as DETR.
3. Analysis Program: The analysis program simplifies and transforms the data into usable patterns. It distinguishes between fixations and saccades and converts them into data that can be utilized.
4. Utilization Program: This program is used to develop assistive features based on eye-tracking. Reference materials like "Eye Tracking in User Experience Design" are consulted to implement functionalities such as fatigue detection and attention monitoring.

How to Design Assistive Features

Text Assistance Functionality:
Determining if Users Need Assistance:
If the gaze duration is excessively long or there are too many instances of re-focusing(Sibert and Jacob (2000)), it indicates that the user may have difficulties reading the text. We can use these criteria to assess whether the user needs assistance in the current environment.

Collecting Concentration Data to Assess User's Focus:
1. Gaze Count: For a single application, the ratio of saccades to fixations can be calculated to determine the user's level of focus.
2. Detection of Blink Duration: When a person is relatively unfocused, the duration from closing the eyes to opening them tends to increase. Therefore, the duration of a single blink can indicate the subject's level of concentration at a given moment (Zhu et al. (2008)).

Total Blinking Time = Average Blink Duration * Average Blink Count
Blinking Behavior Time Ratio = Total Blinking Time / Average Task Time

3.2 YOLOv5 Working Principle

YOLOv5 (You Only Look Once, version 5) is an object detection model developed by Ultralytics. It implements fast and efficient object detection through a single-stage detector. YOLOv5 incorporates various technologies in its design to optimize performance and detection accuracy.

Network Architecture

The network architecture of YOLOv5 primarily consists of the following components:

- **Input Preprocessing**: Input images are first resized to a uniform dimension, such as 640x640 pixels, to fit the network structure.
- **Convolutional Layers**: Multiple convolutional layers are used to extract features from images. Each convolutional layer is usually followed by batch normalization and Leaky ReLU activation functions.
- **Feature Pyramid Network (FPN) and Path Aggregation Network (PAN)**: These integrate both deep and shallow features to enhance the detection capability for both small and large objects.

Anchor Boxes

YOLOv5 utilizes Anchor Boxes to predict bounding boxes. The model learns optimal sizes for anchor boxes during training to accommodate different target sizes.

$$\text{Bounding Box Prediction Formula:} B_x = \sigma(t_x) + c_x \; B_y \quad = \sigma(t_y) + c_y \; B_w = p_w e^{t_w} \; B_h \quad = p_h e^{t_h}$$

where (t_x, t_y, t_w, t_h) are the network's output adjustment parameters, (c_x, c_y) are the coordinates of the top-left corner of the grid cell, (p_w, p_h) are the predefined anchor box width and height, and σ denotes the sigmoid function, ensuring the output ranges between 0 and 1.

Loss Function

YOLOv5's loss function consists of three components: class loss, object loss, and box loss, corresponding to classification, confidence, and bounding box prediction accuracy, respectively.

$$\text{Total Loss} = \lambda_{\text{cls}} \times L_{\text{cls}} + \lambda_{\text{obj}} \times L_{\text{obj}} + \lambda_{\text{box}} \times L_{\text{box}} \tag{3.4}$$

where λ_{cls}, λ_{obj}, and λ_{box} are coefficients that adjust the weight of each part of the loss.

Optimization and Performance

YOLOv5 enhances the training process through strategies such as adaptive learning rate adjustments, data augmentation, and multi-scale training. The commonly used method is the Cosine Annealing Scheduler for learning rate adjustments.(Liu et al. (2023))

Cosine Annealing Scheduler

The Cosine Annealing Scheduler adjusts the learning rate according to the following formula:

$$\eta_t = \eta_{\text{min}} + \frac{1}{2}(\eta_{\text{max}} - \eta_{\text{min}})(1 + \cos(\frac{T_{\text{cur}}}{T_{\text{max}}}\pi)) \tag{3.5}$$

Where:

- η_t is the learning rate at the t-th cycle.
- η_{min} and η_{max} are the minimum and maximum learning rates, respectively.
- T_{cur} is the current cycle number.
- T_{max} is the total number of cycles for learning rate updates, typically set to the total number of training epochs.

This learning rate design begins at its highest value, progressively decreases to a minimum, and then subtly climbs back up. This cyclical pattern assists the model in achieving rapid convergence in the initial stages of training and supports detailed adjustments with a smaller learning rate later on, which helps prevent overfitting.

Impact of Learning Rate Adjustments

YOLOv5 dynamically adjusts its learning rate to meet the varying demands of different training phases. Initially, a higher learning rate is employed to rapidly explore the parameter space. As training progresses, the learning rate is reduced, enabling finer adjustments around the established parameters, which enhances the model' s ability to generalize.

3.3 TFlite and Quantization

In TFLite, quantization is the process of converting a floating-point model into an integer model, aimed at reducing the model size, decreasing memory usage, and increasing inference speed.(Alsing (2018))
The full quantization process employs Quantization-Aware Training (QAT). (Rashidi (2022))The specific steps of quantization are as follows:

1. Define the data types of input and output layers as `int8`。
2. Quantize the features using the following formula:

$$S_3(q_{3_{i,k}} - Z_3) = \sum_{j=1}^{N} S_1(q_{1_{i,j}} - Z_1)S_2(q_{2_{j,k}} - Z_2).$$

3. Quantize the `int32` bias values, where the quantization formula for `bias` is:

$$S_{\text{bias}} = S_1S_2, \quad Z_{\text{bias}} = 0.$$

4. After quantization, apply Batch Normalization to make corrections:

$$w_{\text{fold}} = \frac{\gamma}{\sqrt{EMA(\sigma_B^2) + \epsilon}}.$$

5. After quantizing the `bn` layer, use an activation function such as ReLU. Then, ReLU maintains the input and output as the same data type, i.e., it does not change the data type, continuing to pass data as `int8` .
6. Finally, re-quantize the output to `int8`。

In the above steps, all computations during the quantization process are performed as integers, avoiding floating-point computations and thus enhancing speed and efficiency.(Demosthenous and Vassiliades (2021))

Theoretical Derivation of tflite Quantization Process (int8)

Overview of the model inference process after tflite quantization:

1. The matrix convolution calculation bypasses the de-quantization formula:

$$S_3(q_{3_{(i,k)}} - Z_3) = \sum_{j=1}^{N} S_1(q_{1_{(i,j)}} - Z_1)S_2(q_{2_{(j,k)}} - Z_2)$$

   The formula assumes matrix multiplication between input tensor q_1 and filter kernel q_2, with scaling factors $S1, S2, S3$ compensating for offsets, and zero points $Z1, Z2, Z3$ for alignment of floating-point numbers. The matrix multiplication is defined as $r_3 = r_1r_2$. The relationship between real value(r) and quantization value(q) can be represented by:

$$r = S(q - Z)$$

Expanding the de-quantization formula further yields the output of matrix multiplication:

$$q_{i,k}^3 = Z_3 + M\left(\sum_{j=1}^{N}(q_{i,j}^1 - Z_1)(q_{j,k}^2 - Z_2)\right)$$

Where $M = \frac{S_1 S_2}{S_3}$

Bias is not included in this calculation. Since all calculations, except for M, are within the integer domain, tflite replaces M to minimize the involvement of floating-point numbers and approximate the calculation process as closely to the original values as possible:

$$M = 2^{-n} M_0$$

Here $S1, S2, S3$ are known, and M typically lies within the range $(0, 1)$, allowing the computation of M through bit manipulation of $M_0 \in [0.5, 1]$.

Reducing Matrix Computation Complexity

From the equation:

$$S_3(q_{3_{i,k}} - Z_3) = \sum_{j=1}^{N} S_1(q_{1_{i,j}} - Z_1) S_2(q_{2_{j,k}} - Z_2).$$

To compute this matrix multiplication more efficiently, the 2N3=2NNN subtraction operations involved need complexity reduction. Expanding the original formula yields:

$$q_3^{(i,k)} = Z_3 + M\left(NZ_1Z_2 - Z_1 a_2^{(k)} - Z_2 a_1^{(i)} + \sum_{j=1}^{N} q_1^{(i,j)} q_2^{(j,k)}\right)$$

Where:

$$a_2^{(k)} := \sum_{j=1}^{N} q_2^{(j,k)}, \quad a_1^{(i)} := \sum_{j=1}^{N} q_1^{(i,j)}$$

The final convolution computation related to a1, a2 addition operations has a complexity of $2N^2$. $\sum_{j=1}^{N} q_1^{(i,j)} q_2^{(j,k)}$ still requires $2N^3$ time complexity.

Quantizing int32 Bias: Mathematical Description

Basic Formula

$$S_3(q_{3_{(i,k)}} - Z_3) = \sum_{j=1}^{N} S_1(q_{1_{(i,j)}} - Z_1) S_2(q_{2_{(j,k)}} - Z_2)$$

The prototype of this equation is:

$$r_3 = r_1 \times r_2$$

Add dynamic bias to the quantized value, the new formula is: $r_3 = r_1 \times r_2 + \beta$, where β is the dynamic bias.

From this, we derive:

$$S_3(q_{3_{(i,k)}} - Z_3) = \sum_{j=1}^{N} S_1(q_{1_{(i,j)}} - Z_1) S_2(q_{2_{(j,k)}} - Z_2) + S_{\text{bias}}(\beta - Z_{\text{bias}})$$

To allow bias to share the same scale factor M as the matrix multiplication in formula decomposition, tflite defines the two quantization parameters for bias as:

$$S_{\text{bias}} = S_1 S_2, \quad Z_{\text{bias}} = 0.$$

Thus:

$$q_{3_{(i,k)}} = Z_3 + M \left(\sum_{j=1}^{N} (q_{1_{(i,j)}} - Z_1)(q_{2_{(j,k)}} - Z_2) \right) + \beta$$

Weight and Bias Calculation

$$w_{\text{fold}} := \frac{\gamma w}{\sqrt{\text{EMA}(\sigma_B^2) + \epsilon}}$$

However, if bias is calculated, the computation includes fold bn:

$$y = \gamma_B \left(\left(\sum_{i=1}^{N} w_{i,x_i} + b_{\text{bias}} \right) - \mu_B \right) / \sqrt{\text{EMA}(\sigma_B^2) + \epsilon} + \beta_B$$

Where:

$$w_{\text{fold}} = \frac{\gamma_B w}{\sqrt{\text{EMA}(\sigma_B^2) + \epsilon}}$$
$$b_{\text{bias fold}} = \frac{\gamma_B (b_{\text{bias}} - \mu_B) + \beta_B}{\sqrt{\text{EMA}(\sigma_B^2) + \epsilon}}$$

Fixed-point Approximation of ReLU

After a convolution layer or BN layer that includes a ReLU activation function, interval truncation is incorporated into the corresponding convolution computation. Assuming:

$$S_3(q_{3_{(i,k)}} - Z_3) = \sum_{j=1}^{N} S_1(q_{1_{(i,j)}} - Z_1) S_2(q_{2_{(j,k)}} - Z_2)$$

is followed by a ReLU layer, we get:

$$S_4(q_4 - Z_4) = \text{ReLU}(S_3(q_3 - Z_3))$$

Expressed as a piecewise function:

$$S_4(q_4 - Z_4) = \begin{cases} S_3(q_3 - Z_3), & \text{if } S_3(q_3 - Z_3) > 0 \\ 0, & \text{if } S_3(q_3 - Z_3) \leq 0 \end{cases}$$

Represented as int32:

$$S_4(q_4 - Z_4) = \begin{cases} \text{clip}(S_3(q_3 - Z_3), 0, 255), & \text{if } S_3(q_3 - Z_3) > 0 \\ 0, & \text{if } S_3(q_3 - Z_3) \leq 0 \end{cases}$$

Quantization of the Sigmoid Function

The quantization of the sigmoid function is mainly implemented through the following two steps: The Sigmoid function is defined as:

$$\text{sigmoid}(x) = \frac{1}{1+e^{-x}}$$

We represent it using:

$$D = e^{-x}$$

$$\text{sigmoid}(x) = \frac{1}{1+D}$$

Using the GEMMLOWP library's fixed-point method, we predefine a series of exponent shifts (BARREL SHIFTER) to quickly compute an approximation of e^{-x}:

Exponent	Fixed-point Representation
-2	1672461947
-1	1302514674
0	790015084
1	290630308
2	39332535
3	720401
4	242

Table 3.1: GEMMLOWP EXP_BARREL_SHIFTER Values

Example Results

For instance, for $x = 0.25$, using the formula:

$$e^{-0.25} \approx 0.7788007830714049$$

The corresponding fixed-point representation is $e^{-0.25} \cdot 2^{31} = 1672461947$
$e^{-5.5}$ can be represented as $e^{-4} \cdot e^{-1} = e^{-5} = 0.5$, $e^{-5.1}$ can be represented as $e^{-4} \cdot e^{-1} = e^{-5} = 0.1$。
These common values can be pre-stored in a database. As in the gemmlowp library, the stored constants are as follows:

Exponent	Result	Fixed-point
$e^{-0.25}$	0.7788007830714049	1672461947
$e^{-0.5}$	0.6065306597126334	1302514674
e^{-1}	0.36787944117144233	790015084
e^{-2}	0.1353352832366127	290630308
e^{-4}	0.018315638888734186	39332535
e^{-8}	0.00033546262790251196	720401
e^{-16}	1.125351747192592e-07	242

Table 3.2: Approximation of Exponential Functions and Their Fixed-Point Representations

If $x > -0.25$, which means $|x| < 0.25$, then it is estimated using a fourth-order Taylor expansion. The process is quite intricate, but it also involves step-by-step table lookups.

This quantization method allows for a good approximation of original floating-point numbers in fixed-point integer representation, making it suitable for data processing and computational optimization in various deep learning frameworks.

Chapter 4

Dataset Selection and Annotation for YOLOv5 Model Training

4.1 Dataset Selection

For our project, we accessed various public resources and selected datasets that are suitable for eye movement recognition tasks. Here are the details of the selected datasets:

1. **SCUT-FBP5500 Database**: This dataset contains 5,500 frontal facial images with various attributes (gender, ethnicity, age) and annotations (facial landmarks, beauty scores on a five-point scale, distribution of beauty scores). It supports computational models for predicting facial attractiveness using appearance-based or shape-based methods.

2. **300 Faces In-the-Wild Challenge (300-W), which belong "ICCV 2013"**: Released in 2013, this dataset includes 600 images (300 indoor and 300 outdoor), collected via Google searches using keywords such as "party" and "conference." The images feature significant variations in expressions, lighting, poses, occlusions, and facial sizes. It is annotated with 68 key points, making it suitable for training models that require precise facial feature recognition.

Figure 4.1: SCUT-FBP5500 Database

4.2 Manual Annotation

We installed the Labelimg annotation script on Windows platforms for dataset annotation.
We categorized the dataset based on the direction of the eyeball of a single eye into four classes: up, down, left, right.
Each annotation accurately included the top and bottom edges of the eye socket, and the inner and outer canthi. If an eye in the scene was unrecognizable by human standards, it was not annotated.

4.3 Adding Negative Sample Dataset

Having only annotated positive samples is insufficient for robust model training. We included negative samples to ensure the model can accurately recognize scenes without targets. We collected scenes that are commonly misidentified in eye movement recognition, such as nostrils mistaken for eyeballs and circular objects mistaken as background.
These were added to our dataset without any specific annotations, serving as negative examples.

4.4 Automated Annotation

Using our custom automated YOLO annotation tool, we annotated the data using the model and then adjusted the annotated data. We retrained the data for further learning, thereby achieving reasonably accurate data at a low time cost.
We initially trained a yolov5s model with the above dataset. This model had a certain level of accuracy, but the results were not ideal due to the limited size of the dataset. Consequently, we decided to use this model to batch annotate unlabelled datasets, followed by manual fine-tuning, to generate a large amount of data at low cost.
First, we modified YOLO's 'detect.py' script. We replaced the part of the script that generated annotated images, changing it to save the location information of the targets and create VOC

Figure 4.2: Example:Negative Sample

annotation files named after the images. After running the program, we used Labelimg to adjust the annotations, then converted the VOC format annotation files into YOLO format annotation files. We then merged these datasets with the original dataset and randomly split them into validation sets, using the previously trained model as the initial weights for further training. The process involved the following steps:

1. Train an initial model using a small dataset.
2. Use the initial model to annotate a blank dataset.
3. After annotation, manually check and fine-tune.
4. Incorporate these datasets into the training of a new model.
5. Repeat the above steps.

```
            # Print results
            for c in det[:, -1].unique():
                n = (det[:, -1] == c).sum()  # detections per class
                s += f"{n} {names[int(c)]}{'s' * (n > 1)}, "  # add to string
            for i in range(len(det)):
                # get class name
                class_name = names[int(det[i, -1].cpu())]
                pred_dict = {}
                pred_dict['class'] = class_name
                pred_dict['xmin'] = int(det[i, 0].cpu())
                pred_dict['ymin'] = int(det[i, 1].cpu())
                pred_dict['xmax'] = int(det[i, 2].cpu())
                pred_dict['ymax'] = int(det[i, 3].cpu())
                pred_dict['conf'] = f'{det[i, -2]:.2f}'
                pred_list.append(pred_dict)
            # Write results
            for *xyxy, conf, cls in reversed(det):

                if save_img:  # Add bbox to image
                    c = int(cls)  # integer class
                    label = None if hide_labels else (names[c] if hide_conf else f'{names[c]} {conf:.2f}')
                    annotator.box_label(xyxy, label, color=colors(c, True))
        # Print time (inference-only)
        print(f'{s}Done. ({t3 - t2:.3f}s)')
        create_xml_file(p.name, im0, pred_list, save_dir)
        # Stream results
        im0 = annotator.result()
        # Save results (image with detections)
        if save_img:
            if dataset.mode == 'image':
                cv2.imwrite(save_path, im0)
```

Figure 4.3: Key part code

Chapter 5

Training the YOLOv5 Model

5.1 Environment Setup

We began by downloading Anaconda to manage our model's runtime environment and libraries. We selected the appropriate combination of PyTorch, CUDA, and cuDNN based on our GPU and Python version. This step of setting up the environment is straightforward and does not require detailed description.

5.2 Setting Hyperparameters

Choosing the right batch size is crucial for balancing the efficiency of model training and memory usage. In our setup, we opted for a batch size of 8, which allows us to maintain a fast training speed while reducing memory pressure. We also selected the Adam optimizer due to its excellent performance with non-stationary targets and large datasets, like our eye-tracking dataset.(Warden and Situnayake (2019)) The Adam optimizer combines the benefits of Momentum and RMSprop, automatically adjusting the learning rate to enhance the stability and efficiency of the model training.

5.3 Transfer Learning

Based on the theory of transfer learning, lower-level features are similar across different tasks. We used the yolov5s model provided by the official YOLO implementation as the initial weights for our training. To fully utilize the advantages of transfer learning, we decided to freeze the first three convolutional layers in the early training phase using the command "–freeze 3". This approach allows the model to focus on learning higher-level, task-specific features during training, and speeds up the training process by reducing computational demands.

5.4 Training Process

We imported our annotated training and validation datasets in YOLO format, then executed the training program under the configured Python interpreter. We initiated training with a pre-

trained YOLOv5s model, leveraging the features learned from extensive datasets to accelerate the learning process and improve performance on our specific task. We trained the yolov5s model on the host with settings of batch size 8, model size 320, and 300 epochs. This model was trained using approximately 150 images from the small test set and 30 images from the validation set, all annotated to fit our project's needs from famous open-source datasets like 300-W.

```
(apex8) (apex8) PS E:\APEXAIC\yolov5-master>  python train.py --batch-size 8 --epochs 300 --data D:/fyp/ey
e/mydata_eye_position/data.yaml --weights E:/APEXAIC/yolov5-master/yolov5s.pt --img 320 --freeze 3
```

Figure 5.1: Training Command

5.5 Reviewing Training Results

After training, YOLO saves the trained model and performance charts. The initial training results were not very satisfactory, indicating the need for further training with additional datasets.

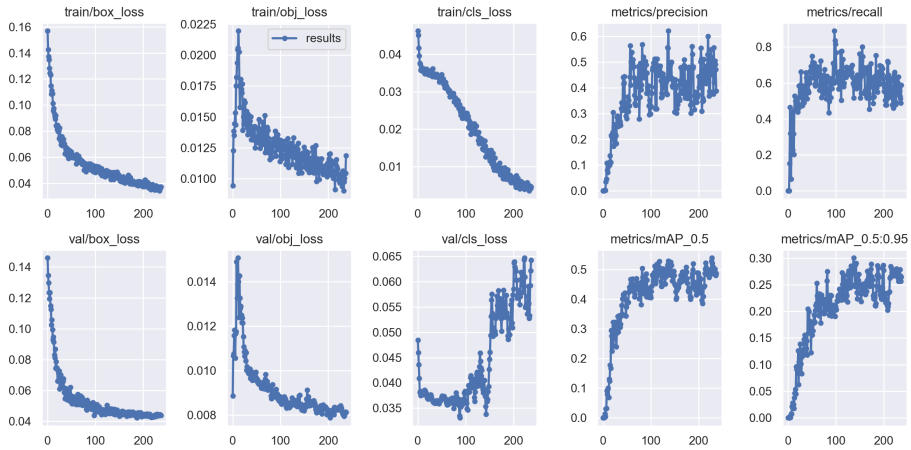

Figure 5.2: Training Results

The training and validation loss trends, precision, recall, and mAP metrics of a model:

Train/Box Loss: This graph illustrates a clear downward trend in the loss associated with bounding box predictions during training. It indicates that the model is becoming more proficient at accurately predicting the locations of objects in images.

Train/Object Loss: This metric represents the model's performance in predicting the presence of objects. The observed decreasing trend suggests continuous improvements over training iterations.

Train/Class Loss: This tracks the loss related to class predictions. A consistent downward trend is evident, signifying that the model is increasingly accurate in classifying objects.

Validation Loss (Box, Object, and Class Loss): These graphs plot the validation loss for boxes, objects, and classes respectively. Both box and object loss exhibit decreasing trends

similar to the training losses, however, the class loss fluctuates more significantly. This might suggest variability in how well the model can generalize its class predictions to new, unseen data.
Metrics/Precision and Recall: These plots show the precision and recall of the model throughout the training iterations. Although both metrics show fluctuations, they generally demonstrate an upward trend, indicating improvements in the model's ability to correctly identify and label objects in the validation set.
Metrics/mAP_0.5 and mAP_0.5:0.95: These metrics represent the Mean Average Precision on the different Intersection over Union (called IoU) thresholds of standard evaluation metric for object detection models. The graphs show variability but generally suggest that the model's prediction accuracy at these IoU thresholds is stable or improving.

5.6 Converting to TFLite Model

The best performing yolov5s model was named "best.pt". We used the official conversion script to transform the pt weight file into a TFLite file, significantly improving inference speed. (Singh and Bhadani (2020))In tests, the inference time for the unconverted pt file was about 400ms, while the converted TFLite file reduced the inference time to approximately 120ms, a notable improvement.

Chapter 6

Android Application Development

6.1 Android Program Operation Process

The operation process of the Android program can be summarized in the following steps:

1. The program captures scene images in real-time through the camera of the Android device.
2. The images are formatted and normalized to meet the input requirements of the deep learning model.
3. Within the neural network, the image data passes through a series of convolutional layers, activation layers (such as ReLU), and batch normalization, among other operations.
4. After processing the image data, the model outputs the results.
5. Tasks are executed based on the output results.

6.2 Creating a New Android Studio Project

After installing and creating a new project, we utilized the TensorFlow official website's Android deployment demo app. We cloned the YOLOv5 official demo using Git tools, but it was outdated and lacked support for some newer operational methods. Consequently, we fetched an optimized version from Maven, where `tensorflow-lite` serves as the core API library for model loading and inference, and `tensorflow-lite-support` provides various common data processing methods. We updated the Gradle version to match the App Demo, installed the Android SDK for the target mobile platform, and set up the necessary debugging software. Due to the poor performance of the emulator and its inability to debug target detection projects, we prepared and connected an Android phone for real-device debugging.
The Android demo provided by TensorFlow significantly reduced our workload. We made modifications to the demo to fulfill our specific needs.

6.3 Improving the Official Demo

First, we replaced the rear camera in the original demo with the front camera and adjusted image parameters, including mirroring the image to ensure that the image data accurately reflects the user's actual eye movements. We then enhanced the app by adding multithreading and GPU delegation features to improve target detection performance. The introduction of multithreading allows the app to simultaneously capture images, process data, and output results, significantly improving efficiency. GPU delegation further optimized the image processing workflow, especially during complex computations and data rendering. Additionally, we considered incorporating NNAPI functionality, which offers hardware acceleration benefits.(Park and Kim (2023)) By utilizing NNAPI (Neural Networks API), we can directly leverage the underlying hardware of Android devices, such as GPUs and DSPs (Digital Signal Processors), thus accelerating inference speed and enhancing efficiency. NNAPI is designed to reduce CPU load, enabling more effective execution of high-performance computing tasks, particularly important for applications requiring rapid response(Ignatov et al. (2021)), such as real-time eye tracking.

6.4 Model Replacement

We replaced the original model with our own trained model and updated the label files. First, we needed to introduce our custom YOLO model, quantified as fp16 with a model size of 320. We then changed the class names to the required categories: up, down, left, right. We also adjusted the input and output vectors according to the actual size of the model to ensure the program runs smoothly.

6.5 Adding New Features

We altered the original layout and introduced new features. First, we removed unnecessary elements, such as the model switching button. We then wrote a detection rule: if the eye looks downward for more than four seconds, the photo gallery is opened. Due to the limited accuracy of the model, it could not trigger reliably. However, despite the limited computing power of my computer, which precluded massive data training, the essential content has already been implemented and only needs further optimization. I believe that with further enhancement and additional features, it can serve as assistive software to facilitate the lives of disabled individuals and also play a role in industrial production.

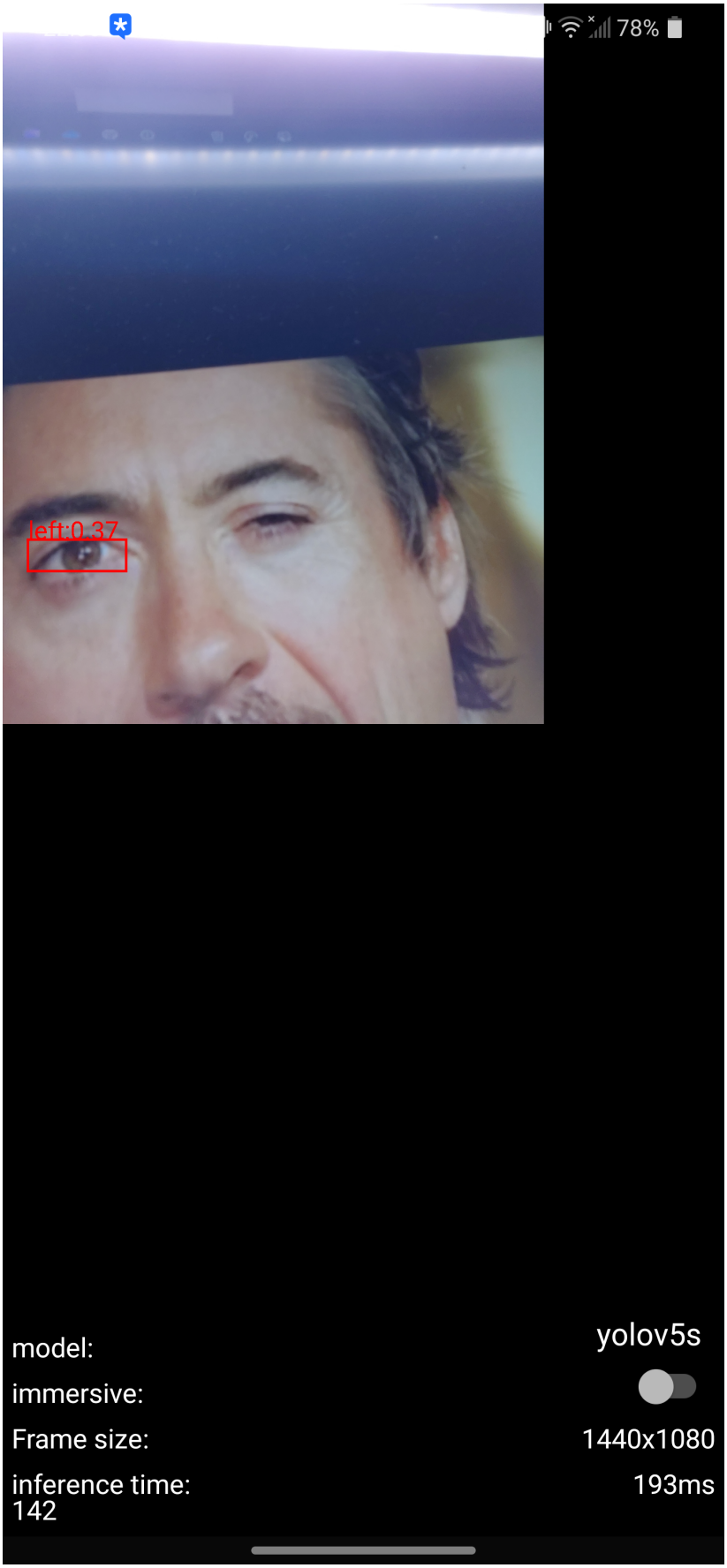

Figure 6.1: App Effect

Chapter 7

Future Scheme

Future Work

1. **Using Side Face Recognition with Deep Residual Identity Mapping for Gaze Point Localization**
   To further enhance the accuracy of eye movement recognition, especially in situations where there is frequent head movement, side face recognition technology combined with deep residual identity mapping (Identity Mapping in Residual Networks)(He et al. (2015)) can be introduced. This method leverages the advantages of Residual Networks (ResNet), using identity mapping to enhance feature transmission and maintain performance stability as network depth increases. By analyzing side-face images, we can more accurately capture the rotation angle of the head and its impact on the direction of eye movement, thus accurately calculating and predicting the actual gaze point in the algorithm. Additionally, the introduction of side-face recognition technology helps stabilize recognition results in complex environments (such as side lighting or partial facial occlusions), enhancing the robustness of the overall system.

2. **Applying NCNN Technology to Optimize Model Performance and Inference Speed**
   NCNN is a streamlined neural network framework designed for forward computation on mobile platforms, particularly effective on devices lacking GPU support. By adapting the YOLOv5 model to the NCNN format, we can greatly diminish its computational load and energy usage while enhancing inference speed. (Rashno et al. (2022))NCNN stands out for its superior memory handling and network optimization, facilitating model compression and acceleration without losing precision.(Kumar et al. (2021)) Additionally, refining and quantizing the YOLOv5 model to eliminate superfluous calculations, coupled with utilizing NCNN, optimizes processing speed and minimizes latency.

3. **Adding More Background Running Features to Enhance User Experience**
   Increasing the app's background running capabilities allows it to continuously monitor eye movements without disturbing the user's normal use of the phone. For instance, a smart monitoring mechanism could be designed to automatically remind users to rest if they have

not changed their gaze direction for a long time, preventing eye fatigue. Alternatively, when users are reading on their phones, the eye-tracking technology could analyze their reading speed and habits to automatically adjust the screen scrolling speed, providing a more comfortable reading experience. Furthermore, background running features could also automatically trigger specific applications or functions when specific eye movement patterns are detected, such as automatically opening emails, social media apps, or activating a voice assistant, thus providing more intelligent and personalized services to users.

Chapter 8

Conclusion

The integration of YOLOv5 with TensorFlow Lite represents a significant enhancement in the field of mobile vision applications, specifically in creating an accessible, efficient eye-tracking system. This research not only confirmed the viability of using YOLOv5 within resource-constrained environments but also showcased how TFLite's model compression and optimization techniques can be utilized to enhance performance without compromising accuracy significantly. The system developed through this study provides a practical tool for real-time eye-tracking, offering potentials for application in user interface accessibility for the disabled, behavioral analysis, and in interactive gaming. Future directions for this research include exploring further efficiency improvements through quantization and pruning techniques in TFLite, and expanding the system's capabilities to include predictive eye movement analytics, which could revolutionize user-device interactions.uture work will aim to further optimize the detection process and expand the application areas of this technology.

Bibliography

Alsing, O. (2018). Mobile object detection using tensorflow lite and transfer learning. *Diva Portal*.

Anliker, J. (1976). Eye movements: On-line measurement, analysis, and control. In Monty, R. and Senders, J., editors, *Eye Movements and Psychological Processes*, pages 185–202. Lawrence Erlbaum, Hillsdale, NJ.

Bergstrom, J. and Schall, A. (2014). *Eye Tracking in User Experience Design*. Elsevier.

Carion, N., Massa, F., Synnaeve, G., Usunier, N., Kirillov, A., and Zagoruyko, S. (2020). End-to-end object detection with transformers. *arXiv preprint arXiv:2005.12872*. Submitted on 26 May 2020 (v1), last revised 28 May 2020 (this version, v3).

Chennamma, H. and Yuan, X. (2013). A survey on eye-gaze tracking techniques. *Indian Journal of Computer Science and Engineering*, 4(5):388–393.

Demosthenous, G. and Vassiliades, V. (2021). Continual learning on the edge with tensorflow lite. *arXiv preprint arXiv:2105.01946*.

Gholami, A., Kim, S., Dong, Z., Yao, Z., Mahoney, M. W., and Keutzer, K. (2021). A survey of quantization methods for efficient neural network inference. *arXiv preprint arXiv:2103.13630*.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*.

Ignatov, A., Byeoung-Su, K., and Timofte, R. (2021). Fast camera image denoising on mobile gpus with deep learning, mobile ai 2021 challenge: Report. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*.

Krafka, K., Khosla, A., Kellnhofer, P., Kannan, H., Bhandarkar, S., Matusik, W., and Torralba, A. (2016). Eye tracking for everyone. *IEEE Computer Society*.

Kumar, A., Vashishtha, G., and Gandhi, C. (2021). Novel convolutional neural network (ncnn) for the diagnosis of bearing defects in rotary machinery. *IEEE Transactions*.

Lee, W. J., Kim, J. H., Shin, Y. U., Hwang, S., and Lim, H. W. (2019). Differences in eye movement range based on age and gaze direction. *Eye*, 33(7):1145–1151.

Leigh, R. and Zee, D. (1991). *The Neurology of Eye Movements*. F.A. Davis, Philadelphia, 2 edition.

Liu, Y., Ping, P., Shi, Q., Chen, H., Yao, Q., and Luo, J. (2023). Research on the cleaning method of unmanned sweeper based on target distribution situation analysis. *Applied Sciences*, 13(23):12544.

Mishra, R., Gupta, H. P., and Dutta, T. (2020). A survey on deep neural network compression: Challenges, overview, and solutions. *arXiv preprint arXiv:2010.03954*.

Park, H. and Kim, S. (2023). *Software Overview for On-Device AI and ML Benchmark in Smartphones*, page Chapter in book. Springer.

Rashidi, M. (2022). Application of tensorflow lite on embedded devices: A hands-on practice of tensorflow model conversion to tensorflow lite model and its deployment. *Diva Portal*.

Rashno, E., Akbari, A., and Nasersharif, B. (2022). Uncertainty handling in convolutional neural networks. *Neural Computing and Applications*.

Seix, C., Veloso, M., and Soler, J. (2012). Towards the validation of a method for quantitative mobile usability testing based on desktop eyetracking. In *Proceedings of the Interaction Conference*, Elche, Alicante, Spain.

Sibert, L. and Jacob, R. (2000). Evaluation of eye gaze interaction. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI 2000, pages 282–288, New York, NY. ACM Press.

Singh, A. and Bhadani, R. (2020). *Mobile Deep Learning with TensorFlow Lite, ML Kit and Flutter*. Packt Publishing.

Ultralytics (2020). Yolov5. `https://github.com/ultralytics/yolov5`.

Warden, P. and Situnayake, D. (2019). *TinyML: Machine learning with TensorFlow Lite on Arduino and ultra-low-power microcontrollers*. O'Reilly Media.

Yarbus, A. (1967). *Eye Movements and Vision*. Plenum Press, New York.

Zhu, Z.-h., Wu, X.-j., and Wang, Lei, e. a. (2008). Detection method of driver fatigue based on blink duration. *Computer Engineering*, 34(5):201–203.