A Smartphone Software Retina

Ryan Wong

School of Computing Science
Sir Alwyn Williams Building
University of Glasgow
G12 8QQ

A dissertation presented in part fulfilment of the requirements of the Degree of Master of Science at The University of Glasgow

September 2017

Abstract

There has been considerable interest in biologically inspired software retinas to produce a data efficient representation of an image. These retinal transformed images can be used as training inputs for deep neural networks for image analysis. Retinal transformed images provide a highly detailed focal area with a lower resolution in the peripheries for point of interests.
Currently, there has been major progress in a desktop software retina but no developments in a mobile version. This work extends on the current implementation of a desktop software retina by porting it to a smartphone application. The smartphone application is built to allow the software retina to automatically capture a large set of areas of interest as a data efficient retinal transformed image. The smartphone software retina is developed to be a portable retina that provides a live camera preview of the retinal transformed images at focal points computed using a gaze control mechanism.
The smartphone software retina was evaluated based on the computational performance of the preparation of the data sets, retinal image transformation computation time and gaze control mechanism's computation time. The results showed that a suitable framework for a smartphone software retina was created which is able to record retinal transformed image data while providing a live preview of these images. Although the smartphone software retina was not able to perform at real-time performance of 30 frames per second, areas of improvement were identified such as preprocessing the preparation data sets. Additionally, a novel approach to a gaze control mechanism was implemented which had minimal affect on the performance of the smartphone software retina's live preview.
The overall result of this work implemented a fully functional smartphone software retina that is able to record retinal transformed image data sets at focal points selected using a gaze control mechanism.

Education Use Consent

I hereby give my permission for this project to be shown to other University of Glasgow students and to be distributed in an electronic format. **Please note that you are under no obligation to sign this declaration, but doing so would help future students.**

Name: _____ Signature: _____

Acknowledgements

I would like to thank my supervisor Dr Jan Paul Siebert for his guidance and support during the development of this project.

I would also like to thank Piotr Ozimek for providing the desktop software retina code and retina tessellation data sets.

Lastly, I would like to thank my family for supporting me throughout my Master's degree.

Contents

Chapter 1

Introduction

This work extends on existing research in biologically inspired software retinas by porting and extending on an existing desktop computer software retina to a smartphone application. This is the first known attempt at the creation of a portable smartphone software retina application that records points of interest in a scene and stores the retinal transformed image data.

Deep learning for image analysis has become a major point of interest due to the advancements of technology and computational power. Many deep learning architectures use the raw image data for training of the deep neural networks. Even with today's technology it requires weeks to months of training. Therefore, there has been a rise in interest in biologically inspired software retinas to create data efficient images to be used as input to the deep neural networks, which helps speed up the training process.

The development of this work focused on porting an existing desktop computer version of the software retina to a smartphone application which is able to provide a live video preview of retinal transformed images. An additional component was added to enable the software retina to capture several points of interest within a scene using a gaze control mechanism.
This work demonstrates a smartphone software retina application that is able to automate the process of capturing retinal image data, by allowing the software retina to transition from one focal point to another using a gaze control mechanism.

1.1 Problem Statement

Existing biologically inspired software retinas are available but are limited to desktop applications. Exploration of a portable software retina is required to produce an automated software retina that records different points of interest captured by the video camera. The mobile software retina must also allow for a live preview of the retinal transformed images for viewing the field-of-view and biological vision. The mobile software retina is an initial stage to gather large retinal imaging data sets required for training in deep neural networks for image analysis / classifications.

1.2 Objectives

The problem statement is broken down into three main objectives / requirements which ensures that a fully functional smartphone software retina is created.

- The first objective is the successful creation of a smartphone application that produces a live preview of the retinal image transforms of each frame captured by the smartphone video camera.
- The second objective is the creation of the gaze control mechanism to find good points of interest with minimal affect on the computational performance of the smartphone software retina, while still allowing for the exploration of the scene captured by the video camera.
- The third objective is the implementation of an effective recording mechanism which is able to store the retinal transformed imaging data for each video frame.

1.3 Motivation

The motivation for this work is to have a portable application which is easy to set up and automatically capture points of interest in an image as a data efficient retinal transformed image. These images can be used for training in a deep neural network. The conventional approach for image processing and analysis is to take an input image and apply feature extraction on the input images and then an action on the features to receive the desired output, as shown in Figure 1.1. The Action stage performs different actions depend on the desired output such as classification, recognition or disparity. The focus of this work with the smartphone software retina is on the stages from the Input Data Set to the Feature Extraction stage.

Figure 1.1: Stages in image processing and analysis.

Recently, the Feature Extraction stage has been removed entirely and the Action stage is applied directly on the raw input image. These image classification / recognition approaches use deep learning with deep neural networks. One of the major drawbacks of this approach, is that a large amount of time is required to process the raw input and therefore training can take weeks to months and requires an extensive number of training items. The use of the retinal transformed images generated from the software retina helps reduce this drawback in that it is data efficient as it stores the relevant and important details in high resolution while the minor details have a lower resolution. Additionally an advantage of using feature extraction and not the raw image for analysis is that the features can be extracted such that considerations can be made for for modality, task, dimensionality, invariance and descriptiveness. The major benefit of feature extraction as opposed to the raw input

image is invariance. Invariance allows for information extracted from the image to be insensitive to changes, which allows a for a easier and faster image classifications and analysis.

Machine vision has been widely successful in finding solutions to specific, well constrained problems such as fingerprint recognition, but biology allows us to find systems that can handle unconstrained, diverse vision problems. Therefore, it is important to understand and use the biological vision system that has been evolved and refined over thousands of years and apply it to modern computer image processing and analysis systems.

1.4 Outline

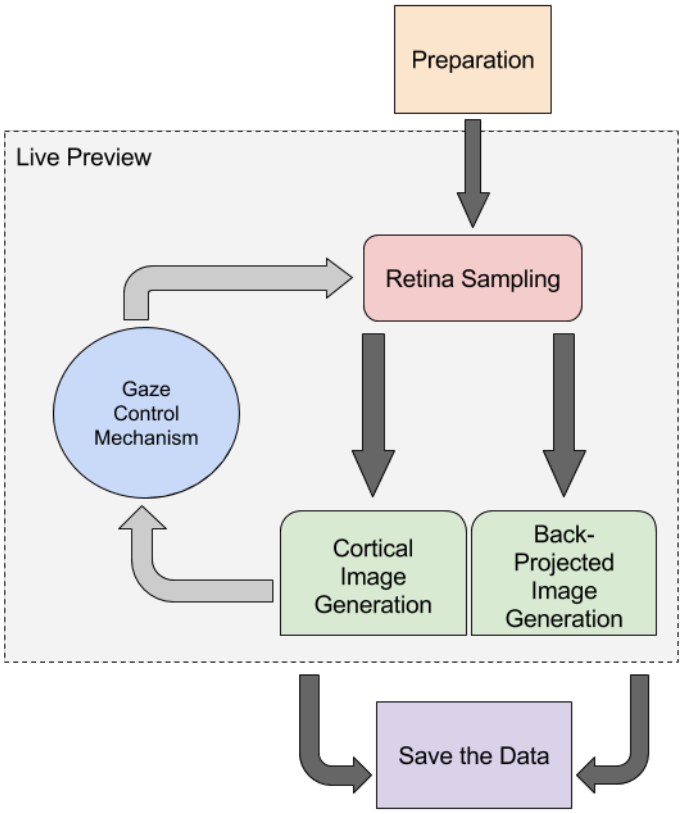

Figure 1.2: Flow diagram of the interaction between stages in the smartphone software retina.

The smartphone software retina undergoes various stages to achieve the goals and objectives identified. Figure 1.2 provides a flow diagram of the life cycle of the smartphone application. These stages are discussed and analysed with an in-depth detailed explanation of the concepts, approaches and evaluation of the implemented smartphone software retina.

The chapters are organised as follows:

- **Chapter 2** contains the background information which highlights key concepts and past work.

- **Chapter 3** explains the approach taken to achieve the objectives set out while highlighting the design, scope and limitations.
- **Chapter 4** contains the implementation and algorithms used to achieve a fully functional smartphone software retina that is able to record retinal transformed image data sets.
- **Chapter 5** contains the analysis and evaluation of the implementation, highlighting the benefits of certain approaches as well as the drawbacks and areas of improvement.
- **Chapter 6** summarises the work, highlighting the achievements and avenues for future work.

Chapter 2

Background

This chapter introduces the relevant concepts of the software retina and discusses the previous work involving software retinas. The human vision system is first explained to give an overview the inspiration behind the software retina and the benefits behind implementing a smartphone software retina. Then a description of existing approaches of a software retina are explained and analysed to determine the best approach for implementing in a smartphone application. Further details of the application of the software retina are examined with past work in gaze control mechanisms to determine how these approaches can be adapted to a mobile application.

2.1 Human Vision System

The human vision system has evolved over thousands of years to reach where it is at now. It is a complex process and contains a range of complex components. This section provides a brief overview of the organisation and stages of the human vision system to provide a background in the core concepts and motivations for a biologically inspired software retina.

The eyeball is stimulated by perceiving light through a hemispherical layer of photoreceptor cells. There are two classical types of photoreceptor cells in the human retina. These are rods and cones which each contribute information to form a representation of the visual world using the visual system. The visual system is part of the central nervous system to process visual details.
Rods are extremely sensitive and are for vision at low light levels with low spatial acuity and do not mediate colour vision [3, 8].
Cones form different functionality and require significantly brighter light as they are capable of colour vision and responsible for high spatial acuity [9]. There are three types of cones which each observe different wavelengths. These wavelengths are similar to the blue, green and red channels in the RGB colour space. The human retina consists of around 6 million cones and 120 million rods [14].

The centre of the retina contains cells which are densely packed called the fovea, which becomes more sparsely distributed in its peripheries [6]. Due to the densely packed fovea, this region is capable of producing the highest visual acuity. The foveated topology of the photoreceptor cells and visual attention mechanisms allow the retina to reduce the amount of information sent to the brain [5]. This is particularly useful as an efficient method for data processing and storage in

the brain such that only the focal area has high visual acuity and the areas around it have lower resolution.
Retina neurons called retinal ganglion cells (RGCs) are responsible for sending the visual signal to the brain. Each RGC is connected to multiple neighbouring photoreceptor cells to form the RGC's receptive fields. The size of the receptive fields increase with distance from the fovea. The RGCs pass the visual information from the retina to the primary visual cortex.

The signal from each eyeball is split into two halves with each half projected separately onto the visual cortex where it is translated to a complex logarithmic mapping which is explained in Section 2.2.1. The mapping to the visual cortex space is said to potentially be a mechanism that allows for scale invariance in the the biological vision system [15].

2.2 Biologically Inspired Software Retina

Over the years there have been various attempts at creating a biologically inspired software retina. One such approach by Balasuriya creates a self-organised software retina [2]. Balasuriya explored the generation, sampling function, feature extraction and gaze control mechanism of the self-organised software retina.

One of the aims of Balasuriya's work was to generate the retina receptive field tessellation. The retina tessellation is the arrangement of the receptive fields. Therefore, the idea was to create a receptive field tessellation which seamlessly changed from a dense uniform fovea to a space-variant periphery of the retina [1]. Wilson showed that there is a significant benefit to the retinal structures as it could extract information that does not change with differences in scale and orientation of an object for a given fixation point [4]. Therefore the retinal structure can be used in implementations of system where objects can be analysed irrespective of its scale and orientation.

The retina tessellations generated by Balasuriya used Clippingdale and Wilsons self-similar neural network model [4]. The retina tessellation employs a self-similar neural network to ensure that there were no local discontinuities, distortions or other artefacts. Each point produced by the self-similar neural network represents a node which defines the location of a receptive field centre. The nodes are used to sample an image at each of their locations. The size of the receptive fields for each node is calculated based on the mean distance to its nearest neighbouring nodes and a scaling factor. The receptive fields follow the biological retinas architecture with each node having a Gaussian response profile with a standard deviation which scales linearly and depends on the local node density.

The generation of the receptive field tessellation used a pseudo-random approach with various parameters affecting the size of the retina tessellation and structure. The parameters are divided into the tessellation parameters and the receptive field parameters. The tessellation parameters included the number of nodes required in the retina tessellation, the fovea radius as a fraction of the tessellation and the number of iterations for the self-organisation of the self-similar neural network used to generate the retina tessellation. The receptive field parameters required the mean pixel distance of five central foveal nodes to their closest five neighbours, a base standard deviation of Gaussian receptive fields and a sigma ratio for the eccentricity scaling factor of the Gaussian receptive fields standard deviation. Figure 2.1 shows an example of a retina tessellation generated with the self-similar neural network model.

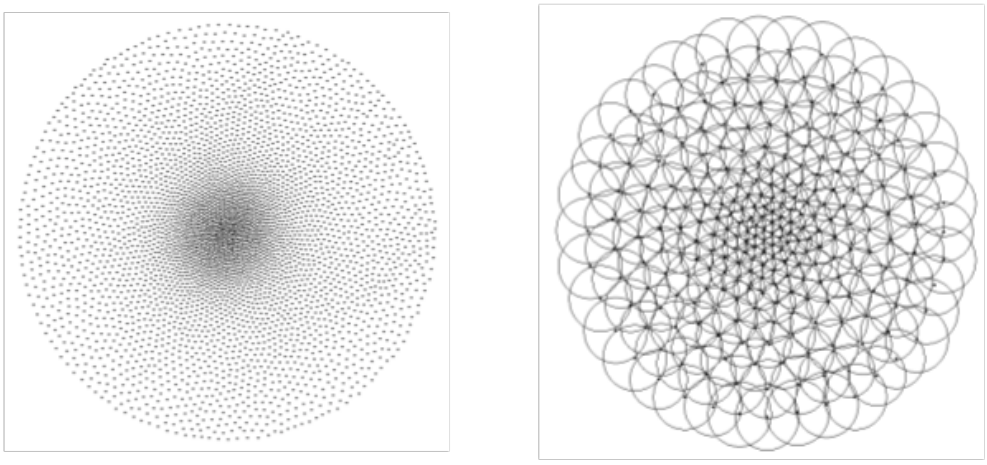

Figure 2.1: An example of the retina tessellation generated by Balasuriya (left) and a retina tessellation generated including the receptive fields for each node (right).

The values sampled on the original image using the receptive fields are stored in an image vector which is a 1D array of intensity values. The image vector is used to create the cortical image and back-projected image which are explained below.

2.2.1 Cortical Image

The cortical image is a depiction of the human vision output generated from the visual system. The cortical image is a data efficient version of the original image at a particular focal point. The cortical image has greater detail in the central area of the retina. The ideal cortical image does not introduce noise and should maintain a fairly uniform receptive field density and preserve all local angles and local information.

Several approaches, using the Gaussian receptive field and the associated image vector intensities, have been proposed to map the original image onto the cortical image space. Schwartz proposed a $log(z)$ model to map from the image space to the cortical space [15]. The $log(z)$ model mapped the original image from Cartesian coordinates (x,y-values) to log-polar coordinates. The log-polar coordinates is the angle about the origin (θ) and the log of the distance from the origin (ρ). The origin is the centre-most point of the fovea.

$$\rho = \log \sqrt{x^2 + y^2}$$
$$\theta = arctan(y/x)$$

The problem with this approach is that a singularity forms at the fovea and there is over-sampling in the foveal region.
The $log(z)$ model was later updated to have a linear polar space model which reduces the effects of severe sparsity in the foveal region and excessive density at peripheries. Therefore the computation of ρ is updated to remove the $\log$ component.

$$\rho = \sqrt{x^2 + y^2}$$

Later, Schwartz created a new $log(z + \alpha)$ model to avoid the singularity at the fovea and reduce the oversampling [16]. The results of the $log(z + \alpha)$ model is a biologically similar cortical image which is split along the vertical half into two visual hemispheres. Since the $log(z + \alpha)$ model splits the cortical image along the vertical meridian into two visual hemispheres the new resultant equations are split between the left and right half of the cortical image.

$$\begin{aligned} X_{left} &= -\sqrt{(x-\alpha)^2 + y^2} \\ Y_{left} &= arctan(y/(x-\alpha)) - sign(arctan(y/(x-\alpha))) * \pi \\ X_{right} &= \sqrt{(x+\alpha)^2 + y^2} \\ Y_{right} &= arctan(y/(x+\alpha)) \end{aligned}$$

The cortical images are produced by projecting Gaussian matrices scaled by the associated image vector values on the appropriate node locations. The two halves of the cortical image are realigned through rotations and scaling. The resulting cortical image does not introduce any noise or artefacts and has the added benefit of preserving local angles, sufficiently uniform density receptive field projections and has preserved local information captured by the retina.

The cortical image is the main image of interest in the smartphone software retina as it is a data efficient method to storing images while preserving the details for areas of interest. The cortical images are the input images which can be used as the training data for deep learning.

2.2.2 Back-Projected Image Generation

The back-projected image is not required for storage purposes and mainly functions as a visual aid. The back-projected image allows for visibly easier checks to be conducted on the image to determine the sharpness and artefacts created by the retinal sampling. It can also be used to check the location of the focal point and the field-of-view within the original image space.

The back-projected image is generated by projecting every Gaussian receptive field onto the original image plane and scaling it by the corresponding image vector values. The resulting image is then normalised using a pre-generated Gaussian normalisation of the original image. The final result is the back-projected image which only has the details in the retina's field-of-view within the image space. Therefore, the back-projected image has a focused high resolution centre (fovea) and lower resolution peripheries. Since the back-projected image is projected onto the original image space, the back-projected image is the same resolution as the original image but the image can be reduced by cropping it to only include the area containing the retina's field-of-view.

2.3 Gaze Control Mechanisms

Eye movements are classified in several ways. One such classification of eye movement is through gaze-shifting or visual fixation. Gaze-shifting can either be saccade movements or smooth pursuit eye movements. Saccade eye movements are quick movements between fixation points in the same direction, while in smooth pursuit movements the eyes move smoothly to follow a point of interest instead of in jumps [10]. In this work, focus is mainly conducted on saccade movements / saccade targeting. The reason for saccade targeting as opposed to smooth pursuit eye movement is because saccade movements allow for instant recording of different point of interests without needing to follow a point from one location to the next location and allows for faster computation since it is an instant movement to the desired location.

Balasuriya used saccade targeting as a means for gaze control [2]. The interest points detected were based on the space-invariant visual information extracted by the self-organised artificial retina. The interest points used matching based on the Hough transform and required a large number of complex computations not suitable for run-time application. Balasuriya used the interest points along with an attention and saliency mechanism to determine the next focal point. Figure 2.2 shows the flow diagram for Balasuriya's feed-forward model for space-variant vision and saccade generation.

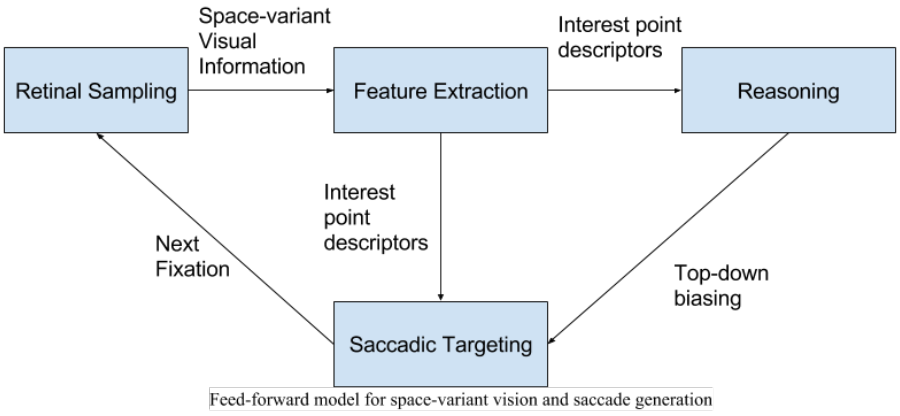

Figure 2.2: Flow diagram of Balasuriya's feed-forward model for space-variant vision and saccade generation.

Ozimek implemented a rudimentary gaze control implementation [12]. The idea behind the approach taken was to maximise the likelihood of the retina being fixated upon salient areas of the image. This approach is similar to Balasuriyas approach in that they both maintain a saliency map of the input image. The saliency map is populated using the retinal back-projected image for finding SIFT features. SIFT is scale-invariant feature transform which is an algorithm to detect and describe local features in images [11]. Once all the SIFT features are found a Gaussian is projected at each feature's corresponding location in a special saliency map. The fixation points are represented in the inhibition-of-return map as an amplified Gaussian spanning the retinas foveal region. The next focal point is found by subtracting the inhibition-of-return map from the saliency map and selecting the coordinate with the maximum value on the final saliency map.

In both approaches, the authors used either the original image or the back-projected image for determining the points of interest which requires processing of the whole image which could prove

to be time consuming for real-time applications.

2.4 Desktop Software Retina

A current implementation of Balasuriya's software retina is available in Python. The Python implementation used OpenCV as its core library for handling the image manipulation and image transformations. OpenCV is a cross-platform library containing programming functions which are able to perform real-time computer vision algorithms [13]. The desktop software retina is able to perform retinal image transforms on a live preview captured by the computer's webcam.
The software retina takes the retina tessellation data sets generated in a Matlab program and transforms the video frame to either a cortical image or back-projected image. However, the desktop software retina is limited to a focal point in the centre of the image. The live preview does not have any gaze control mechanism and therefore cannot find points of interest. The desktop software retina also does not record the cortical data sets for each image and only provides the visual images. Therefore, new algorithms and approaches are required to handle these issues but the algorithms need to have limited affect on the performance of the live preview of the retinal transformed image generation.

Overall the Python implementation for Balasuriya's software retina provides the algorithms and general tools required to perform the retinal transform of the images at a set central focal point from a live video camera.

2.5 iOS Applications

iOS is an operating system for Apple Inc.'s mobile phones. Development for iOS applications use iOS Software Development Kit (SDK) which supports the programming languages Swift and Objective-C. iOS is the second most popular mobile operating system in the world while Android is the most popular.
iOS applications follow a model-view-controller architecture. A model-view-controller architecture means that the application's development is separated by the application's data, logical components and the visual presentation [7]. This is important for the generalisation and compatibility of iOS applications between different devices. One of the major benefits for developing for iOS devices is that there are very few different models of phones which use iOS and therefore it is easier to make applications cross compatible with multiple devices. This differs to Android where there are hundreds of devices, each with different device specifications such as screen sizes and camera settings. iOS application development is chosen as this works development focus due to the availability of an iPhone device and that it is easier to program an application that runs on multiple devices without issues. Even though the Apple Developer Account's initial process with signing applications is difficult set-up, once completed the applications can be run, tested and deployed easily.

2.6 Overview of Software Retinas

In this chapter, an overview of the concepts and reasoning behind a software retina is given, where a cortical image provides greater data efficiency that using the raw image data. Additionally, the cortical image provides a degree of invariance which is extremely useful for image analysis, classification and reasoning. Previous work has shown considerable improvements in algorithms and mappings from the image space to the cortical space. Although there has been exploration in gaze control mechanisms for finding points of interest, the performance of these approaches may not perform well during real-time computations. The gaze control mechanisms analysed are required to be modified and made simpler to have minimal affect on the live preview, while still ensuring that good points of interest are found. Points of interest are considered good if they encourage exploration of the image space and are unique points which stand out in the image.

The work of Balasuriya's software retina approach provides a suitable starting ground for the creation of a smartphone software retina. In the next chapter, the approach to porting existing code to a smartphone application is discussed with the limitations and scope of this work.

Chapter 3

Approach and Design

This chapter gives an overview of the approach to create the smartphone software retina. A brief description of the smartphone application is given with languages and software tools used while highlighting areas which have caused problems and the solutions to these problems. Several constraints, constants and limitations are also identified to ensure the application is supported across multiple devices and allows for accurate evaluation of the implementation.

3.1 Porting to the Mobile Application

There is no known previous attempts at creating a smartphone software retina and previous work mainly focused on a desktop software retina. The desktop software retina used to port the code to a smartphone application used Balasuriya's algorithms in a Python application which used the OpenCV libraries.

The implementation in this work was limited to iOS smartphone applications. The code was implemented in Objective-C. OpenCV with Objective-C has been well documented and tested when compared to the latest Swift programming language which is a programming language for iOS that is also supported by Apple Inc. Objective-C allows for easier porting from Python as they are both object-oriented programming language and both are compatible with OpenCV. The initial set-up for preparing Objective-C programs with OpenCV requires initialisation of certain parameters in the Xcode project file. Appendix B.1 provides the details for this initial set-up process.

The two main areas of focus and concern for the smartphone software retina are the retinal image transforms and the gaze control mechanism.

3.1.1 Retinal Image Transforms

For the generation of the cortical and back-projected images, the performance of the initial setup and considerations of the limitations of the smartphone application need to be acknowledged. Current implementations of the retina algorithms require libraries that are not available on smartphone applications and the Python algorithms require a redesign to support Objective-C's data structures,

which may reduce the performance of the software retina. The set-up time for the application must also be fast, as the application is designed for portability and easy recording of the retinal transformed image data. The priority of the smartphone software retina is therefore aimed to be computationally efficient. Therefore the algorithms and implementation discussed in Chapter 4 use computational times as a basis for the evaluation.

3.1.2 Gaze Control Mechanism

The gaze control mechanism requires a new approach to identify points of interest as it needs to be computed in real-time. The approach taken in this work uses the cortical image as the method to identify points of interest. Therefore the focal point will follow a saccade movement but will still find interest points which are within the previous field-of-view. This removes the effect of the retina jumping from one side of the image to the other side. Further details of implementation and algorithms for the gaze control mechanism is explained in Section 4.1.5. The general idea is to allow the gaze control system to use distance as well as key-point responses to identify the next point of interest while ensuring that there is continuous movement of the retina and wide exploration of the scene. Saccade targeting also allows the retina to record multiple different points of interest while still remaining computationally fast such that it has minimal affect in the run-time of the smartphone software retina.

3.2 Application Design and Features

The smartphone application is designed to have 4 different modes. Each mode provides a different image view. The four modes are highlighted as follows:

- **(Original) Image Space Mode** - This mode provides a view of the video camera output without any processing or translations. It contains the full resolution of the image in the scene captured by the camera.
- **Cortical Image Space Mode** - This mode provides a view of the cortical image which is created from the image in the original image space from a specific focal point.
- **Back-Projected Image Space Mode** - This mode provides a view of the back-projected image which has the same resolution as the original image.
- **Combined Images Mode** - This mode provides a view of both the back-projected image and the cortical image side-by-side. The gaze control mechanism is also implemented in this mode so that the user is able to view of the field-of-view and focal point of the retina in the back-projected image with the cortical image without needing to change modes.

Figure 3.1 shows the output view the smartphone application for each of the modes discussed. Additionally, there is an option to save the recording of the retinal data sets required to recreate the cortical image while in the Combined Images Mode. The retinal data starts recording once the user enters the Combined Images Mode. The details of what and how the data is saved is explained in Section 4.1.6.

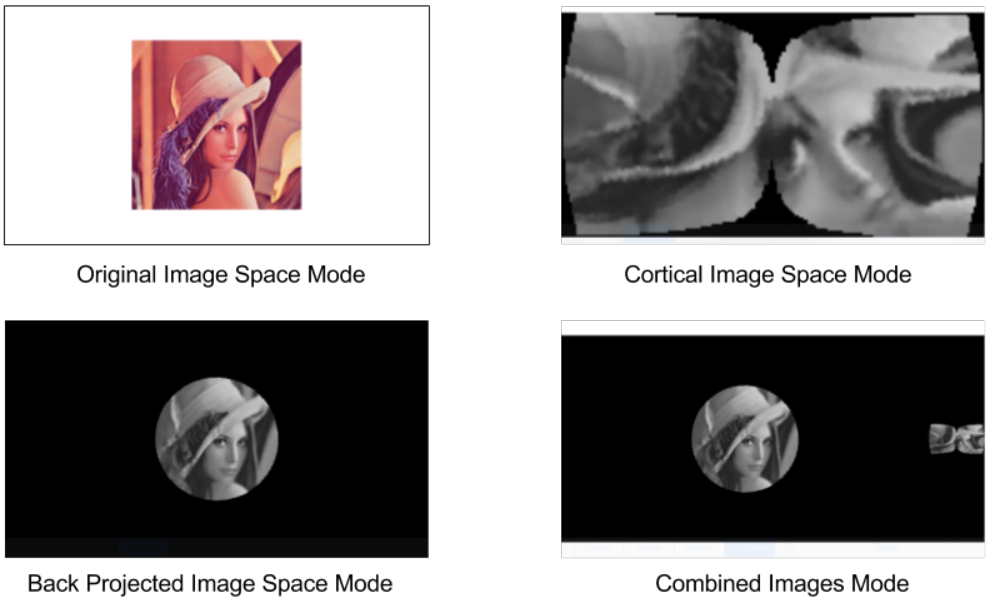

Figure 3.1: View of each mode on the smartphone software retina application.

3.3 Constants and Limitations

Several constant values and data sets are used in the implementation of the smartphone software retina. In the following section each of these constants are discussed with reasons for the particular choices. Although this work mainly focused on the use of certain data sets and variables, the implementation is not limited to these constants and can be applied to other values and data sets with the same data structure.

The first constant data set used is the retina tessellation. In this work, the largest retina created at the time is used as it can be used to extract the most data in an image. The retina tessellation has 8192 receptive fields. The retina tessellation is generated using the self-similar neural network model by Balasuriya.
The retina tessellation data is stored in two data sets. The first data set contains the receptive field locations (`rf_loc`) and the second contains the receptive field coefficients (`rf_coeff`). In the following subsections the two data sets are discussed where each receptive field is called a node.

3.3.1 Receptive Field Locations

The data structure for the `rf_loc` data set contains the details required to determine the node's location. Each node contains a list of 7 floating point values. The 3rd, 4th, 5th and 6th value for each node are all initialised to 0. The 1st, 2nd and 7th value contain the location and receptive field size details for each node.

The first and second values for each node contain the x and y Cartesian coordinates respectively. The coordinates contain the location of the node's centre based around the origin which is the centre of the fovea. The seventh value for each node is the total width of the receptive field. Therefore a

node within the fovea has a smaller receptive field width than a node at the peripheries. Figure 3.2 depicts the data structure of a node in the `rf_loc` data set generated using the self-similar neural network model and a diagram showing the location of the node in the retina tessellation.

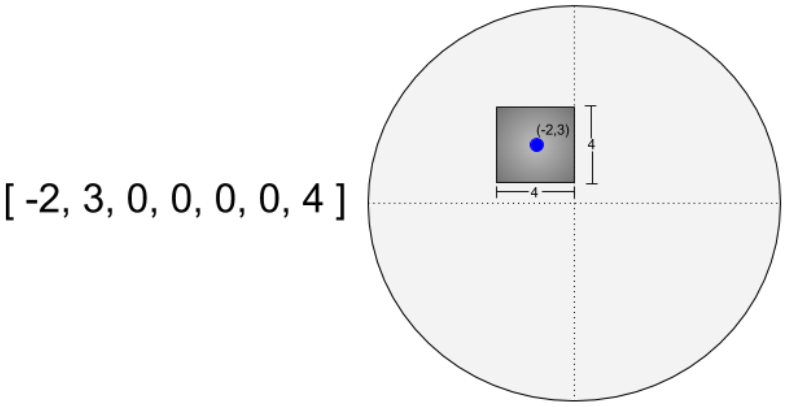

Figure 3.2: Representation of the data structure of a node in the `rf_loc` data set and the location of the node in the retina tessellation.

3.3.2 Receptive Field Coefficients

The data structure for the `rf_coeff` data set contains the details for the coefficients of the matrix for each node's receptive field. Therefore the size of the matrix for the nodes match the 7^{th} value in the `rf_loc` data set. The coefficients are stored as floating point numbers in a square matrix. In Figure 3.2 the node's receptive field width is 4 which means the coefficient matrix is a 4x4 matrix.

In both `rf_loc` and `rf_coeff` data sets, the nodes are sorted from closest to the fovea centre to the furthest away. Therefore matching nodes in `rf_loc` and `rf_coeff` has the same index in both data sets.

3.3.3 Other Constraints and Limitations

Other constraints included setting the camera resolution to 1280×720 to ensure that the software retina application is compatible across different iOS devices. The resolution of 1280×720 was used as the retina tessellation of 8192 nodes was large enough to include detail for points of interest but still allow freedom of movement for the focal points without frequently reaching the borders of the image.

The approach taken for the processing of the software retina used only grayscale images as the current system of the software retina on the desktop applications could not perform at real-time with coloured images. Therefore grayscale images are used on the mobile application to ensure that the software retina is able to provide a live preview of the cortical and back-projected images. The colour space images may be possible to use as an extension to this work by using threads or mobile GPU optimisations.

3.4 Overview of the Approach and Design

In this chapter, a description of the design, scope and limitations of the smartphone software retina were discussed. Prerequisites for the implementation were also explained with reasoning for using certain software tools and data sets. The next chapter provides an in-depth explanation of the implementation with the description of the stages involved in the smartphone software retina. The stages involved in the smartphone software retina are used to produce the different images for the view modes. The screenshots of the full design of the software retina application are shown in Appendix B.2.

Chapter 4

Implementation

This chapter discusses the implementation to address the problem statement and meet the objectives identified for the smartphone software retina. The implementation of the software retina is broken down into several stages which each address a core component of the system. These stages are the preparation, retina sampling, cortical image generation, back-project image generation, gaze control mechanism and saving the data. Lastly, an overview of the interactions between the stages are given with the applications of the smartphone software retina such as for recording cortical data for training into deep neural networks.

4.1 Stages of Software Retina

The development and implementation of the smartphone software retina is broken down into several key stages. The stages include preparation, retina sampling, cortical image generation, back-projected image generation, gaze control mechanism and saving the data. These stages help separate core components of the implementation of the smartphone software retina. The following subsections outline the algorithms and the implementation for each of these stages.

4.1.1 Preparation

The preparation stage involves computing all the necessary data sets before performing the operations with the live video frames from the camera. The preparation stage uses the retina tessellation location data set (`rf_loc`) to compute the mapping from the image space to the cortical space. The mapping is computed during the Preparation stage to reduce the computations required during the live preview. Additionally, other preprocessed variables are computed such as the Gaussian kernels which are used during the Cortical Image Generation stage discussed in Section 4.1.3.
The preparation is completed in three sub-stages discussed below.

Splitting the retina tessellation nodes

The first component involves splitting the node locations into the left and right halves in the image space and updating the `rf_loc` data set.
The 3rd value of each node in the `rf_loc` data set is updated to contain the index of the node, Equation 4.1 shows the new updated data structure for a node in `rf_loc`.

$$[x, y, index, 0, 0, 0, width] \tag{4.1}$$

The two new data sets created split the node locations in `rf_loc` to left node locations (`leftrf_loc`) and right node locations (`rightrf_loc`). The left and right nodes are split by the centre vertical line crossing the origin of the retina tessellation. The data structure of these data sets contain three floating point numbers for each node as shown in Equation 4.2.

$$[x, y, V_{index}] \tag{4.2}$$

The x and y values are the coordinates for nodes in the retina tessellation while the the V_{index} contains the index for the node based on its index in `rf_loc`. Algorithm 1 depicts the pseudocode to generate the new data sets and update the existing `rf_loc` data set.

Data: `rf_loc`
Result: `leftrf_loc` and `rightrf_loc`
for *node in* `rf_loc` **do**
  UPDATE $node[2]$ to $node$'s index
  if $node[0]<0$ **then**
    APPEND $node[0:3]$ to `leftrf_loc`
  else
    APPEND $node[0:3]$ to `rightrf_loc`
  end
end

Algorithm 1: Pseudocode to generate the separate halves of the retina tessellation data sets and update `rf_loc`.

Mapping from the Image Space to Cortical Space

The next phase in the Preparation stage involves mapping from the data sets `leftrf_loc` and `rightrf_loc` image space coordinate values to the cortical space. The algorithm used to map the coordinates to the cortical space is the improved $log(z + \alpha)$ model described in Section 2.2.
The resulting data sets are the cortical space coordinates for the left nodes (`leftcort_loc`) and right nodes (`rightcort_loc`). The data structure for each node in `leftcort_loc` and `rightcort_loc` contains two floating point numbers which represent the x and y coordinates respectively. Unlike the coordinates in `leftrf_loc` and `rightrf_loc`, the coordinates in `leftcort_loc` and `rightcort_loc` are based on image matrix coordinates and not Cartesian plane coordinates, which makes the data sets easier and faster to work with when completing the

image manipulations. Figure 4.1 depicts an example of how the coordinate system works for the cortical space location data sets.

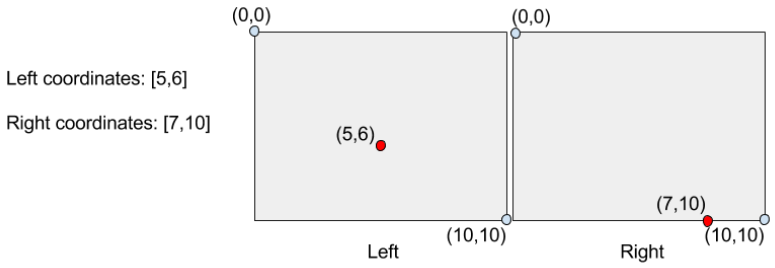

Figure 4.1: Representation of the coordinate structure for `leftcort_loc` and `rightcort_loc` data set.

Adjusting the cortex

The final phase involved in the Preparation stage involves adjusting the cortical locations (`leftcort_loc` and `rightcort_loc`) to fit within the image boundaries with simple boundary checks. The cortical image also undergoes scaling to reduce the jaggy edges created due to the mapping from the image space to the cortical space. The implementation in this work used a scaling factor of 0.5 as it showed to reduce the jaggy edges in the cortical images. The Gaussian kernel matrix (`G`) is also generated in this phase. The Gaussian kernel matrix is a 10×10 matrix with a chosen kernel size of 7×7. The Gaussian kernel matrix is used in the Cortical Image Generation stage for the normalisation of each of the node's receptive fields.

Implementation of the Preparation Stage

There are two approaches to create these data sets. The first approach is to complete all the preparation components on the mobile application at start up. This approach involves porting the original algorithms from Balasuriya's work to Objective-C for iOS. This type of approach is referred to in future sections as the online approach, as it is completed on the device.
The second approach is to complete the Preparation stage on a desktop program via preprocessing and save it to files stored on the mobile application. This type of approach is referred to in future sections as the offline approach. The files are read on the application's start-up and the data sets are saved to their relevant variables. This approach aims to reduce the preprocessing time on start-up to ensure the smartphone application can be used with minimal delay.
The online and offline approaches both have several advantages and drawbacks which are analysed and evaluated in Chapter 5.

4.1.2 Retina Sampling

The Retina Sampling stage extracts the space-variant visual information from the image space based on a fixation point. The approach is similar to the biological vision system where the feature extraction operates on a specific limited spatial receptive field and not the whole field-of-view. The

output data set generated by the retina sampling stage is a one dimensional image vector (`IV`). The image vector is used to generate the cortical image and back-projected image which are discussed in the next sub-sections. The image vector needs to be calculated for each video frame as it requires the original image frame and the focal point location. Algorithm 2 shows the calculation for the image vector by using the receptive field's region of interest for each node and multiplying it by the node's `rf_coeff`.

Data: `rf_loc`, `rf_coeff`, focal point (x,y), img
Result: `IV`
for $node$ *in* `rf_loc` **do**
  $x1 \leftarrow node[0] + x - node[6]/2$
  $y1 \leftarrow node[1] + y - node[6]/2$
  $x2 \leftarrow node[0] + x + node[6]/2$
  $y2 \leftarrow node[1] + y + node[6]/2$

  $ROI \leftarrow img[x1 : x2, y1 : y2]$
  $V[node$'s index] $\leftarrow$ SUM($ROI \times$ rf_coeff[$node$'s index])
end
Algorithm 2: Pseudocode to calculate the image vector for a particular image frame.

4.1.3 Cortical Image Generation

The cortical image is generated by computing each half of the cortical image separately. Each half undergoes a rotation before being combined where the left half undergoes a rotation by 90 degrees anticlockwise and the right half rotates by 90 degrees clockwise. Figure 4.2 shows an example of the transformation from image space to cortical space.

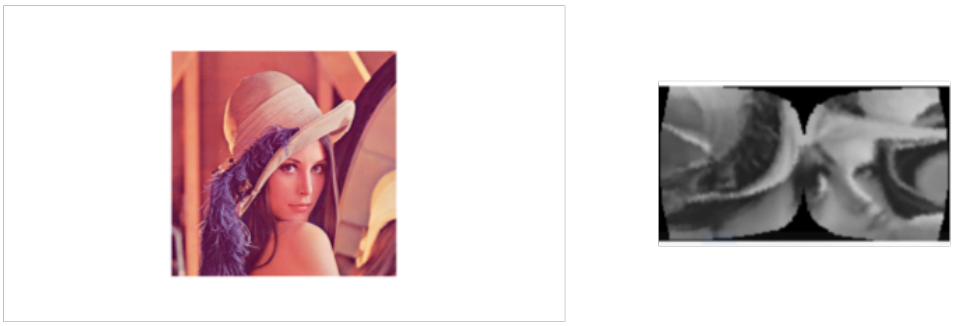

Figure 4.2: An example of the retinal transform from the image space (left) to the cortical space (right).

The cortical image for each half is calculated by determining the sum of the product of the Gaussian filter array matrix and the image vector for each node. This is then normalised according to the sum of the Gaussian filter array for the nodes. Algorithm 3 describes the process to generate the left half of the cortical image `leftcort`. The right half of the cortical image (`rightcort`) is computed with the same method but the right retina tessellation data sets.

Data: `IV`, `leftrf_loc`, `leftcort_loc`, `G`
Result: `leftcort`
for *node in* `leftcort_loc` **do**
    $xCorner1 \leftarrow node[0] - (G\text{'s width})/2$
    $xCorner2 \leftarrow node[0] + (G\text{'s width})/2$
    $yCorner1 \leftarrow node[1] - (G\text{'s width})/2$
    $yCorner2 \leftarrow node[1] + (G\text{'s width})/2$

    $dx \leftarrow$ first decimal place value for leftcort_loc[0]
    $dy \leftarrow$ first decimal place value for leftcort_loc[1]

    $ROI_g \leftarrow G[dx, dy][gy1 : gy2, gx1 : gx2]$
    $L_{img}[y1 : y2, x1 : x2] += g * IV$[leftrf_loc[2]]
    $L_{gimg}[y1 : y2, x1 : x2] += g$
end
leftcort $\leftarrow L_{img}/L_{gimg}$

Algorithm 3: Pseudo-code to calculate `leftcort`. The same process can be applied for `rightcort` but with the right half's data sets.

The resulting cortical image is a highly reduced, data efficient image when compared to the original image as it preserves high resolution detail of the areas in and around the point of interest and has lower resolution in the peripheries.

4.1.4 Back-Projected Image Generation

The back-projected image is used mainly for visualisation purposes as it is easier to view the area of focus in this image space as opposed to the cortical image. It is calculated by going through the receptive field locations and recreates the retina vision in the original image space. Figure 4.3 shows an example of the view of the transformation from the original image to the back-projected image.

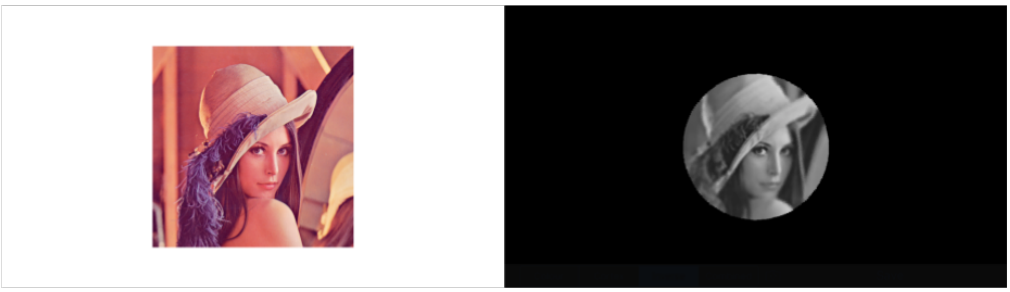

Figure 4.3: An example of the view of the transformation from the original image (left) to the back-projected image (right).

Algorithm 4 describes the process where each receptive field is updated in the image space by

the cumulative sum of each node's image vector value multiplied by the node's `rf_coeff`. The resulting output is the back-projected image (`bp_img`). The Gaussian normalisation of the image (`GI`) is calculated just before the generating the back-projected image. The Gaussian normalisation cannot be computed during the Preparation stage like in the desktop software retina, as the focal point changes for each frame.

Data: `IV`, focal point (x, y), `rf_coeff`, `rf_loc`, `GI`
Result: `bp_img`
for *node in* `rf_loc` **do**
  $x1 \leftarrow node[0] - (GI$'s width$)/2$
  $x2 \leftarrow node[0] + (GI$'s width$)/2$
  $y1 \leftarrow node[1] - (GI$'s width$)/2$
  $y2 \leftarrow node[1] + (GI$'s width$)/2$

  $I[x1 : x2, y1 : y2] += IV[node$'s index$] \times$ rfcoeff$[node$'s index$]$
end
bp_img $\leftarrow I/GI$

Algorithm 4: Pseudocode to calculate the back-projected image `bp_img`.

4.1.5 Gaze Control Mechanism

The gaze control mechanism in this work applies a novel approach to reduce the effect on the computational performance of the live preview in the software retina when selecting a new focal points. The approach uses the cortical image as the method for finding focal points based on key-points identified with the SIFT key-point detector. The cortical image is used to find the key-points as it has a much smaller resolution than the original image space which speeds up the process of determining focal points. Additionally, this approach is similar to a biological approach in that the cortical space is used to find focal points and therefore the retina can only find points of interest within its field-of-view. The SIFT key-point detector is used as it has the advantage of having efficient performance and can find individual features which are distinct.

The approach takes 50 key-points found with the SIFT key-point detector after applying a mask to the cortical image to ensure that only the points within the retina vision is used. Figure 4.4 shows the mask applied to the cortical image. The SIFT key-point detector is only applied to the white area of the mask.

Figure 4.4: The mask applied to the cortical image to find SIFT key-points

The key-points are organised into respective quadrants according to their location on the Cartesian plane with the centre of the cortical image as the origin. The reason for organising the key-points into quadrants is to allow for the gaze control mechanism to choose the focal point with high probability to move in similar directions to previous directions. The choice of the gaze direction is based on the four quadrants and is updated depending on the probabilities listed below:

- 50% of the time the gaze direction is kept in the same direction
- 20% of the time the gaze direction moves to an adjacent quadrant
- 20% of the time the gaze direction moves to the other adjacent quadrant
- 10% of the time the gaze direction moves in the opposite direction

Therefore the gaze direction will try to follow the same or similar direction to the direction it was moving before. This allows for smooth and continuous focal point movement from one frame to the next frame and reduces the chances of the focal point oscillating between two focal points which improves the exploration of the retina. Figure 4.5 depicts an example case where the original gaze direction is in the first quadrant. The next gaze direction is then chosen based on the probabilities mentioned above. Therefore, there is a 50% chance to keep the gaze direction in the first quadrant, 20% chance for either the second quadrant or fourth quadrant and 10% to move choose a focal point in the third quadrant.

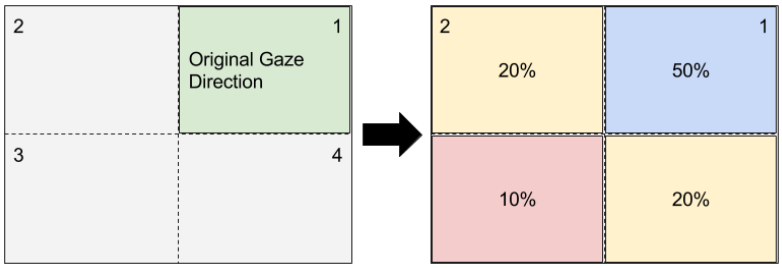

Figure 4.5: An example case of gaze control where the original gaze direction is in the first quadrant. The right diagram shows the probabilities of the next quadrants being chosen.

The gaze directions is not the only factor that affects the choice of the key-point used. The key-points are also sorted according two factors.
The first factor is the horizontal distance from the vertical centre of the cortical image. A distance measure is used to keep the focal point from staying in the same or similar places and allows the retina to explore other areas of the image. Therefore, the further the key-point is from the vertical centre, the more likely it will be chosen as the next focal point. The horizontal distance is used as it is fast to determine and a good approximation to the real distance in the image space. The distance is divided by the half the cortical image width to normalise the number between 0 and 1.
The SIFT key-point response value is the second factor affecting the choice of key-points. The response value gives an indication of how much the key-points stands out from the image. It can be viewed as the strength of the key-point. Therefore the higher the response value, the more likely the key-point is a good point of interest.
These two factors are multiplied together for each key-point to create the new key-point rank (kp_{rank}) and sorted by the rank from highest to lowest. Equation 4.3 shows how the key-point

ranking is calculated. The idea behind this approach is to choose a combination of the best strength of the key-point and the furthest key-point from the origin.

$$kp_{rank} = \frac{(kp_{loc} - origin_{loc})}{(cort_{width}/2)} \times kp_{response} \tag{4.3}$$

Once the best key-point is chosen based on the correct gaze direction and highest key-point ranking, the key-point location in the cortical space is mapped back to the image space. Previous work did not require the individual coordinates to map back to the image space and therefore the implementation required a new inverse algorithm. The mapping from the cortical space to the image space is completed by computing the inverse of the equations discussed in Section 2.2.1. The inverse mapping cannot directly be calculated from the equations and required preprocessed data values. These data values contained the scaling factor constants and receptive field location minimums in the image space. The resulting output gives the new focal point which is used to calculate the new image vector for the new cortical image and back-projected image.

4.1.6 Saving the Data

Since the smartphone application requires a live preview, every time a new video frame is updated with a new focal point, it requires every frame of the original image to be saved. In order to save less data the image vector is also saved and not the full cortical image. The image vector and the original image can be used offline on a desktop computer to generate the cortical image using the same method discussed in the Cortical Image Generation stage. Along with the image vector the focal point is also required to be save to determine the centre of the retina tessellation in the image space. The image vector can also be used to generate the back-projected image.
Therefore, the required data sets saved are the image vectors, associated focal point coordinates and the original image frames captured. For the smartphone application, the recording of the data is completed automatically using the Gaze Control Mechanism and Retina Sampling Stages. The user indicates when to stop recording the data and the information is saved to a file on the smartphone.

4.2 Overview of the Software Retina

The smartphone software retina uses all the stages mentioned in the previous section to create an automatic retina data recording tool which also provides a live preview of the retina vision system. Figure 1.2 shows a flow diagram of how each of the stages interact with each other to produce a functioning live smartphone software retina that is able to record the cortical image data sets. The resulting data from recording the cortical image data set can then be used to recreate the cortical images and used in applications such a training data in deep neural networks for image classification or object detection. As discussed in Section 3.2, the smartphone application has four different modes. These modes use the relevant stages required for generating the desired output. An example of this is if the Cortical Image Space mode is selected then the stages of the smartphone software retina follow from the Preparation stage to the Retinal Sampling stage to the Cortical Image Generation Stage. The Back-Projected Image Space Mode undergoes the stages from the Preparation to the Retinal Sampling to the Back-Projected Image Generation stage. The Combined

Images mode is similar but include the gaze control mechanism and the option to save the recording of the retinal data.

Chapter 5

Evaluation and Analysis

This chapter gives the details of the evaluation process and analysis of the performance and effectiveness of implementation. The evaluation is conducted under several constraints to ensure accurate analysis is conducted on the results. The evaluation is conducted on three areas of the smartphone software retina. These areas are the preparation computation time, retina image transformation time and gaze control focal point computation time. Lastly, analysis of additional observations are discussed and areas of improvement for the smartphone software retina.

5.1 Evaluation Constants

As discussed in Section 3.3, several constant data sets are used for the implementation of the smartphone software retina. These constants are also used for the evaluation of the software retina. The device used to test the performance is a iPhone 5C which has the latest, at the time of evaluation, operating system of iOS 10.3 installed. The iPhone 5C has a 8-megapixel rear camera, 1GB of RAM and the Apple A6 processor which is a 1.3GHz dual-core processor.

5.2 Verification and Validation

The verification and validation of the implementation of the smartphone software retina used unit testing as a method of testing whether the desired results were obtained at various stages of the retina. The results of the unit tests were compared to the results of the desktop version of the software retina. The results of the smartphone implementation showed to match those of the desktop implementation. Though it was found that there was a precision error in the Python desktop version where some floating point variables were not typecasted correctly to integer values. This was fixed in the smartphone implementation. The method to fix the typecasting error involved rounding the floating point values first before typecasting to integers.

5.3 Preparation Data Set Generation Evaluation

The preparation data set generation involves the Preparation stage discussed in Section 4.1.1 which computes the relevant data sets before beginning the live preview of the software retina. The following data sets are generated during the preparation stage:

- `rf_loc` - receptive field node locations
- `rf_coeff` - receptive field node coefficients
- `leftrf_loc` and `rightrf_loc` - receptive field node locations split by cortical halves
- `leftcort_loc` and `rightcort_loc` - receptive field node locations in the cortical space split by cortical halves
- `G` - Gaussian kernel matrix

These data sets are evaluated on the computational time it takes to generate, which is therefore the time it takes the smartphone application to go from a start-up state to the live preview state. The two approaches of offline and online data generation are tested independently for five runs with the results for the computation time measured in seconds as shown in Appendix A.1.

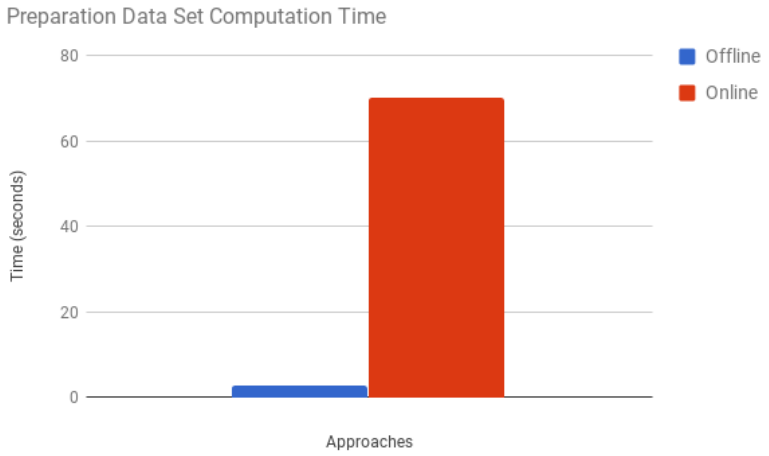

Figure 5.1: Graph representing the computational time to generate the preparation data sets.

Figure 5.1 shows the average computational time to generate the preparation data set for both the offline and online approaches. The average computation time for the offline approach was 2.94 seconds while the online approach took on average 70.22 seconds. This is a drastic difference in the computational performance. The offline approach performs significantly faster which is extremely useful for mobile applications where you can open the application and almost immediately start recording retina data. One potential problem is that the preparation data sets need to be computed on a computer first and then transferred to the mobile application. The online approach may be better for when multiple retina tessellations are used in quick succession but it has one major drawback in that every time the application is opened it would need to compute the preparation data sets. An attempt was conducted to try compute the preparation data sets at the first start-up and save it but

there were issues of the requirement of hard-coded values due to the combination of OpenCV and Objective-C data type conflicts.

On analysis of the online approach, it was found that the algorithm which significantly increased the computation time of the data sets is the function which computes Euclidean distance between each pair of the two collections of inputs ($cdist$ function). The $cdist$ function is computed for all nodes in the retina tessellation at four different areas of the preparation stage. On the smartphone application there were no built in libraries for Objective-C of such an algorithm and it was computed with an inefficient performance of $\mathcal{O}(n^2)$. Algorithm 5 shows the $cdist$ algorithmic process.

Data: array
Result: resultArr
for i *in length(array)* **do**
  for j *in length(array) starting at* $j = i$ **do**
    $resultArr[i][j] \leftarrow \text{DIST}_{eucl}(array[i], array[j])$
  end
end

Algorithm 5: Pseudocode for the $cdist$ function.

The offline approach which used a desktop computer to compute the preparation data sets could use external libraries which included the an optimised $cdist$ function and therefore not take as long to compute. Therefore exploration of libraries or alternative methods to compute the $cdist$ function may be useful to further improve the online approach to possibly compete with the offline approach's computational time.

5.4 Retinal Image Transformation Evaluation

The performance of the live preview is based on the computational time required to generate the cortical and back-projected image. A combination of the two images is also evaluated as it is part of the major component of observation for the live preview especially for monitoring the gaze control's focal point and field-of-view. Figure 5.2 shows an example of the two images combined. The cortical image looks significantly smaller than the back-projected image as they are on the same scale. Therefore the back-projected image has a resolution of 1280×720 and the cortical image has a resolution of 194×103.

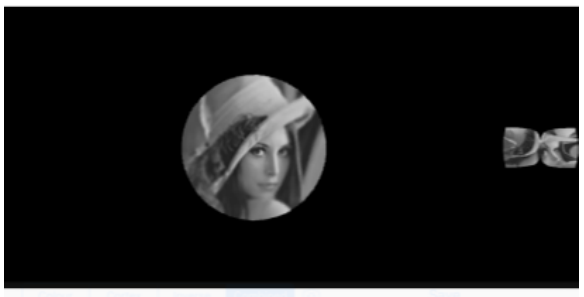

Figure 5.2: An example of the view of the combined images of the back-projected and cortical image.

The tests are completed in a controlled system where each view mode is generated independently over 10 runs. Appendix A.2 shows the results for each run for the different image view modes. The focal point also remains in the centre of the image space to ensure that there are no external factors affecting the speed of the image generation such as the gaze control mechanism. Figure 5.3 shows the resulting computation times for each of the output images.

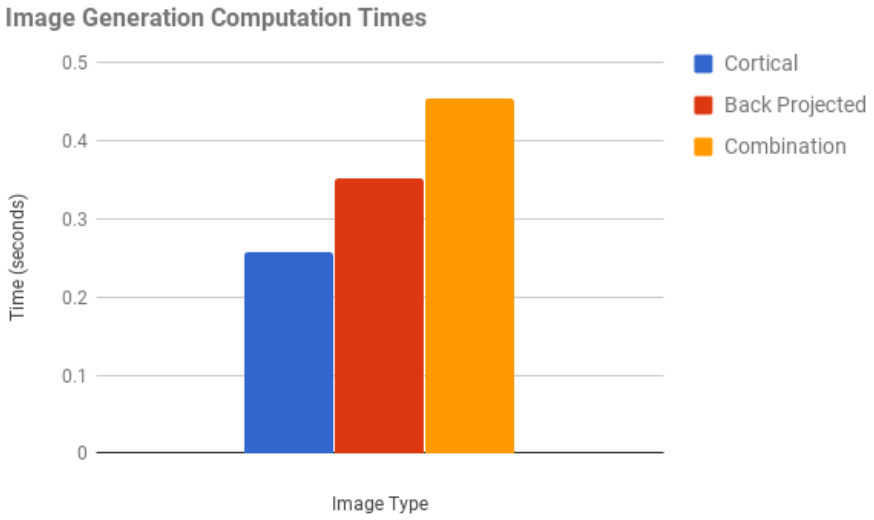

Figure 5.3: Graph representing the computational time to generate the cortical, back-projected and combined images.

The average time to generate the cortical image was the lowest at 0.257 seconds, next the back-projected image took 0.351 seconds to generate and the combined image took 0.455 seconds. The times to generate the images can be converted to frames per second (FPS) but none of the images generated are close to the desired real-time performance of 30 FPS. The cortical image generated achieved the highest FPS of around 3.9 FPS. The cortical image generation was 0.094 seconds faster to generate than the back-projected image as the cortical image resolution was significantly smaller than the back-projected image. The generation of the back-projected image can be optimised by cropping the image to only the area within the retina's field of view but it would make it difficult to distinguish where the retina's focal point has moved. The combination of cortical and back-projected image did rather poorly and only achieved 2.2 FPS as it needed to compute both images and combine them into one image.

In all the above times the retina sampling stage was included as part of the image generation. The retina sampling was also tested independently and it was found to take on average 0.13 seconds to complete. Although this time seems insignificant, it has a major effect on the performance of the generation of the image. An example of this is that the retina sampling uses 51% of the computation time to generate the cortical image. Therefore not only is important to find improvements in generating the retinal transformed images but also the retina sampling stage. For future work, the smartphone software retina's performance can be improved by adapting the algorithms to use multithreading or a mobile GPU to speed up the algorithms during run-time.

5.5 Gaze Control Mechanism Performance Evaluation

The gaze control mechanism is evaluated based on the computational time it takes to find the new focal point. The idea behind measuring the computational time of finding the new focal point is to ensure that the gaze control mechanism has a minimal affect on the performance of the software retina. The computation times for the focal points are completed for 20 new focal points. Figure 5.4 shows the computation times for calculating the new focal points. The computation times are based on the gaze control mechanism discussed in Section 4.1.5, where the output focal point is the coordinates in the original image space. The resulting mean time for computing the new focal point is 0.064 ± 0.007 seconds.

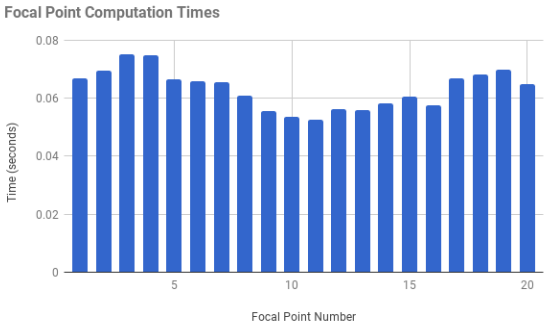

Figure 5.4: Graph representing the computational times to find a new focal point.

The benefits of this approach is that it only uses the cortical image to find the key-points. The focal point computation times have a minimal affect on the live preview retinal image transform generation as the inversion from the cortical space back to the image space is completed in $\mathcal{O}(1)$. There is a drawback to using the cortical image as it limits exploration especially if only one object is in the field-of-view. This can be overcome by applying random resets or random movements to the gaze control mechanism. Alternatively a mapping of areas already explored can be used to determine the value of going to certain areas of the image space. The algorithms implemented in the gaze control mechanism has the flexibility to include these additional customisations. The implementation of the gaze control mechanism also allows for adjustments in how the focal point is chosen as each component is calculated independently. The following components can be switched out for different approaches or algorithms as long as the output data structure of each component remains the same.

1. **Key-point detector** - The implemented key-point detector in this work used the SIFT detector.
2. **Gaze direction algorithm** - The gaze direction in this work used a Cartesian quadrant system which can be switched for other directional heuristics or exploration mappings.
3. **Key-point ranking** - The key-points in this work were ranked according to the multiplicative value of the distance from the origin and SIFT key-point response values.
4. **Inversion algorithm** - The inversion algorithm applied in this work is the most accurate approach to get a point from the cortical space to the image space but alternative approaches are possible such as look-up tables.

Overall the gaze control mechanism implemented allows for the flexibility of exploration of alternative algorithms or parameters. Another benefit of this approach is that if the device is mounted onto a robotic arm then the gaze control system can also be applied to the movement of the arm. Therefore instead of moving the focal point in the image, the device can can be moved on the robotic arm and not be limited by the image boundaries.

5.6 Overall Evaluation

The overall smartphone software retina is able to record retinal data from the camera's video stream. Although the software retina was not able to provide a real-time smooth retinal transformed image, the implementation provided a suitable framework for future work in exploration of optimisations in the image generation and focal point exploration. Figure 5.5 shows an example of the some preview image outputs generated at when using an image with the gaze control mechanism implemented.

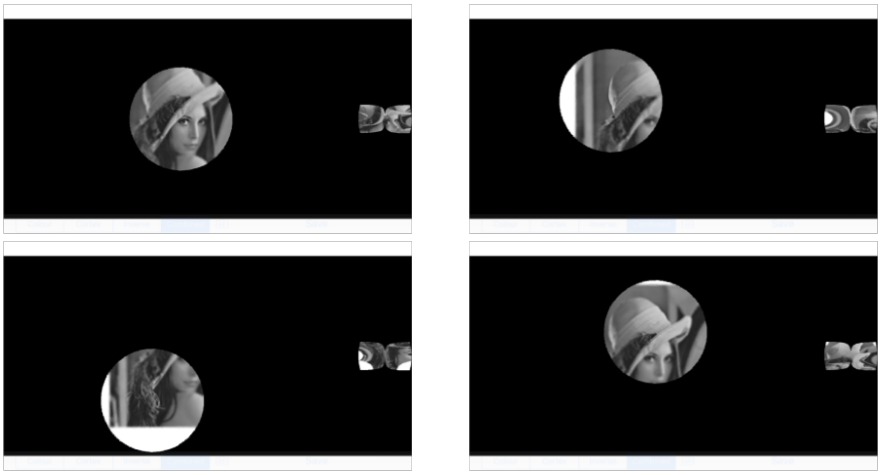

Figure 5.5: Figure of a few focal points in the Lena Image

One problem identified with the implementation of the software smartphone retina is that the amount of data saved is limited by the device storage capabilities and copying the image data sets also showed to have an impact in the speed of the live preview. The file size of the data recorded grows at a rapid rate due the need to store the large original image data for each frame which is a matrix of size 1280×720. This issue can be resolved by cropping the image to only contain the field of view of the retina tessellation but then it is not possible to obtain the full image to monitor the gaze direction. A potential solution to this problem is to store and upload the retinal image data sets on a cloud service and therefore not be limited by the devices storage.

In this chapter, the evaluation and analysis found that the implementation can be used as a suitable framework for future smartphone software retina. Areas of improvement have been identified and with the approaches taken, it is possible to adapt and modify the code to handle these improvements.

Chapter 6

Conclusion

The primary motivation for the implementation of a software retina on a mobile device was to investigate an effective approach that is able to run a live preview of the retinal image transforms while automatically capturing points of interest from the video camera. In this chapter, an overview of achievements are given. Additionally, avenues for future work are identified and some final remarks of the overall smartphone software retina.

6.1 Achievements

The objectives identified in Section 1.2 have been fully achieved in this implementation of the smartphone software retina. In this section, a list of the achievements are identified based on the implementation and evaluations.

- **A successful approach to porting an existing Python implementation of the software retina to an iOS application in Objective-C was achieved.**
  The iOS application is able to provide a live preview of different modes of the retinal image transform. The code from the Python implementation was successfully ported such that the same cortical and back-projected transformations were applied in the smartphone application. Additionally, the smartphone application included a mode where both cortical and back-projected images appeared side-by-side, which is used with the gaze control mechanism.

- **An area of improvement was identified whereby an offline preparation approach was implemented to reduce the time required for generating the data sets before the live preview is shown.**
  The direct porting of the desktop software retina to the smartphone application was a challenging task as the preparation data sets took a significantly long time and was required to be computed every time the application was opened. This reduced the effectiveness of a portable software retina. An alternative approach was implemented where the preparation data sets were completed offline and saved onto the mobile application. This significantly improved the speed in the start-up process. The drawback to this approach is that every time a new retina tessellation is used, then the data files need to be recreated on the desktop application and then uploaded to the application.

- **This work introduced a flexible and fast novel approach to a gaze control mechanism for finding points of interest in a image using the data efficient cortical image.**
  The gaze control mechanism implemented used several features to determine points of interest. The features depended on a directional bias, distance measure and SIFT key-point responses. The gaze control mechanism has a significant advantage where it has multiple independent methods and can easily be adapted and modified as the current implementation lacked exploration of the full image.
- **A recording option of the retinal transformed image data created by the live preview was successfully implemented to save the data to the device.**
  The smartphone software retina is able to save the original image, image vector and location of the focal point for each frame. These data sets can be used to recreate the cortical images which can be used as training in a deep neural network. The approach taken showed to be effective but was limited by the device's storage capabilities and can be improved with compression algorithms and saving the data in cloud storage instead.

6.2 Future Work

The approach and implementation can be used as a framework for any mobile application and improved in various ways. This section highlights some of the key areas for future work.

6.2.1 General Improvements

Some general improves in performance include the use of multi-threading and the mobile GPU to speed up computations. Additionally, currently the system only supports grayscale, an improvement could be to allow for colour version. In this work a general approach to a smartphone software retina is described, the approach taken used the iOS operating system but future work could include using the same approach in the Android operating system. Building the iOS applications requires a large amount of time for setting up developer accounts and has a high cost for distributing applications. Android may be more suitable for allowing easier distribution of the application. The one major drawback is that there are many different Android devices and the development of the application must take device compatibilities into account.

6.2.2 Gaze Control

The gaze control system can be updated to handle exploration using an inhibition-of-return map by exploring areas that have not been visited before. Other areas of improvement are through updating the key-point ranking method to use other calculations that may aid exploration and find strong points of interest. When updating the gaze control mechanism it is important to take into account how the algorithms affect the performance of the live preview as the software retina is aimed at providing real-time retinal image transforms.
Additionally tests of the gaze control mechanism can be conducted on the mobile mounted on a robot which will also move as the gaze control moves, therefore it will not be limited by the image space captured by the camera.

6.2.3 Saving the Data

The saving component implemented in this work used a general approach which has several areas of improvement. The first area of improvement is via compression whereby currently the data saved does not have any form of compression for the data. Secondly, currently the data is saved to the mobile application and requires a connection to a computer to extract the data, an improvement to this can be by using a cloud service to save the data online. This will thereby not limit the size of storage to the device but the cloud storage capabilities.

6.3 Final Remarks

This work has ported the original software retina to a mobile application with modifications for improved performance via preprocessing. A novel approach to gaze control was also introduced and used for finding points of interest which was then used for recording the cortical image data. This is a first approach to a portable system for retrieving cortical image data for training in deep learning. The next stage in the overall bigger picture and motivation of the software retina involves testing the system for recording the large quantities of data and evaluating the effectiveness in deep learning architectures. Other applications include using the smartphone software retina to control robots or mounting the device onto a robotic arm for observations and movement. The applications of a smartphone software retina can be applied to various tasks and has great potential for future exploration.

Appendix A

Table Appendix

A.1 Preparation Data Set Computation Time

	Offline (seconds)	Online (seconds)
1	2.950406	70.029762
2	2.918871	70.311429
3	2.937027	69.871096
4	2.928058	70.349811
5	2.990117	70.536295
Average	2.9448958	70.2196786

Table A.1: Preparation Data Set Computation Time

A.2 Image Generation Computation Times

	Cortical (seconds)	Back Proj (seconds)	Combined (seconds)
1	0.263149	0.334455	0.45197
2	0.24791	0.333976	0.448325
3	0.26152	0.337469	0.445752
4	0.262643	0.346021	0.470209
5	0.263893	0.363897	0.455744
6	0.265052	0.353614	0.455837
7	0.251897	0.343264	0.451263
8	0.24756	0.364962	0.464979
9	0.248138	0.381823	0.460223
10	0.261566	0.353233	0.441189
Average	0.2573328	0.3512714	0.4545491

Table A.2: Image Generation Computation Times (seconds)

A.3 Focal Point Computation Times

Number	Time (seconds)
1	0.066824
2	0.069636
3	0.075105
4	0.074958
5	0.066414
6	0.065879
7	0.065664
8	0.060815
9	0.055576
10	0.053501
11	0.052627
12	0.056152
13	0.055993
14	0.058127
15	0.060553
16	0.057639
17	0.066941
18	0.068070
19	0.069859
20	0.064894
Average	0.064261

Table A.3: Focal Point Computation Times

Appendix B

Design and Approach Appendix

B.1 Xcode Project File Parameters

1. Setting up environment
   - Add files to project → $AppProjectPath$/opencv2.framework
   - File → Add File to $ProjectName$ → Supporting files folder for $images$
2. Configure the project
   - $ProjectName$ → General → Deployment info
     - check - Hide status bar
     - check - Requires full screen
   - Info.plist file
     - add ”View Controller-based status bar appearance” and set to ”NO”
     - add ”video-camera” to ”Required device capabilities”
     - add ”Privacy - Photo Library Usage Description” and set to ”$(PRODUCT_NAME) photo use”
     - add ”Privacy - Camera Usage Description” and set to ”$(PRODUCT_NAME) camera use”
   - $ProjectName$ → Build Phases → Link Binary With Libraries
     - add ”Accelerate.framework”
     - add ”AssetsLibrary.framework”
     - add ”CoreGraphics.framework”
     - add ”CoreMedia.framework”
     - add ”CoreVideo.framework”
     - add ”Photos.framework”
     - add ”Social.framework”
     - add ”UIKit.framework”
     - check that ”opencv2.framework” is already in the list, add it if it is not in the list

- $ProjectName \rightarrow$ Build Settings
  - Apple LLVM $no.$ - Language
    - $*$ change compile source to "Compile Source as Objective C++"
  - Apple LLVM $no.$ - Preprocessing $\rightarrow$ Preprocessor Macro
    - $*$ set Debug to "WITH_OPENCV_CONTRIB DEBUG=1"
    - $*$ set Release to "WITH_OPENCV_CONTRIB Any Architecture - Any SDK"

B.2 Screenshots of the Application Design

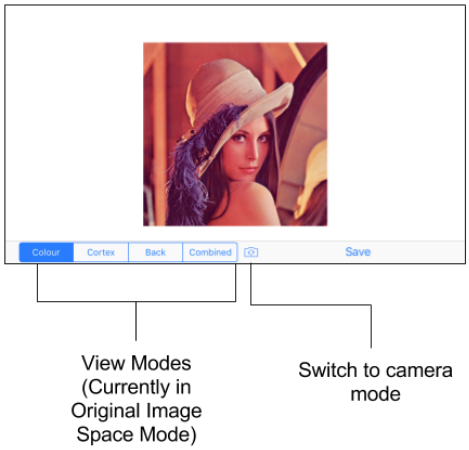

Figure B.1: Screenshot of application in Original Image Space Mode, the video camera option will display the live video recording instead of the defaulted still image.

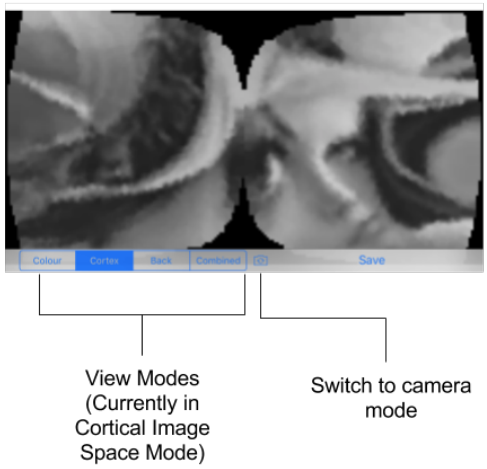

Figure B.2: Screenshot of application in Cortical Image Space Mode, the video camera option will display the live video recording of the cortical image generated.

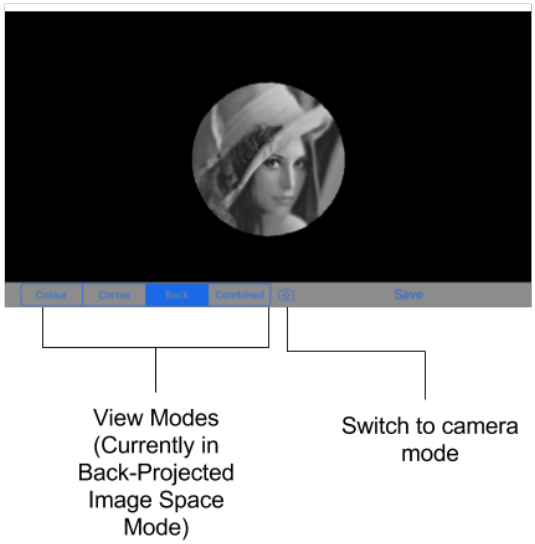

Figure B.3: Screenshot of application in Back-Projected Image Space Mode, the video camera option will display the live video recording of the back-projected image generated.

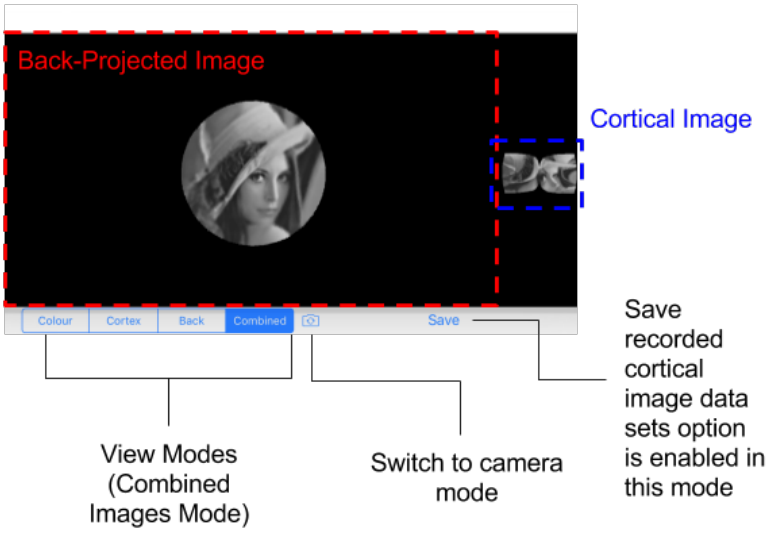

Figure B.4: Screenshot of application in Combined Images Mode, the video camera option will display the live video recording of the both the back-projected image and cortical image. The save option will save the cortical data sets to a text file on device's application and can be accessed via connecting the phone to a computer.

Bibliography

[1] LS Balasuriya and JP Siebert. An artificial retina with a self-organised retinal receptive field tessellation. In *Biologically-inspired Machine Vision, Theory and Application Symposium*, 2003.

[2] Sumitha Balasuriya. *A computational model of space-variant vision based on a self-organised artificial retina tessellation*. PhD thesis, University of Glasgow, 2006.

[3] DA Baylor, TD Lamb, and King-Wai Yau. Responses of retinal rods to single photons. *The Journal of physiology*, 288(1):613–634, 1979.

[4] Simon Clippingdale and Roland Wilson. Self-similar neural networks based on a kohonen learning rule. *Neural Networks*, 9(5):747–763, 1996.

[5] Christine A Curcio and Kimberly A Allen. Topography of ganglion cells in human retina. *Journal of comparative Neurology*, 300(1):5–25, 1990.

[6] Christine A Curcio, Kenneth R Sloan, Robert E Kalina, and Anita E Hendrickson. Human photoreceptor topography. *Journal of comparative neurology*, 292(4):497–523, 1990.

[7] Mark H Goadrich and Michael P Rogers. Smart smartphone development: ios versus android. In *Proceedings of the 42nd ACM technical symposium on Computer science education*, pages 607–612. ACM, 2011.

[8] Selig Hecht, Simon Shlaer, and Maurice Henri Pirenne. Energy, quanta, and vision. *The Journal of general physiology*, 25(6):819–840, 1942.

[9] Leo M Hurvich. Color vision. 1981.

[10] Richard J Krauzlis. The control of voluntary eye movements: new perspectives. *The Neuroscientist*, 11(2):124–137, 2005.

[11] David G Lowe. Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, volume 2, pages 1150–1157. Ieee, 1999.

[12] Piotr Ozimek. Integrating a biologically inspired software retina with convolutional neural networks. 2017.

[13] Kari Pulli, Anatoly Baksheev, Kirill Kornyakov, and Victor Eruhimov. Real-time computer vision with opencv. *Communications of the ACM*, 55(6):61–69, 2012.

[14] Daniel L Schacter, Daniel T Gilbert, and Daniel M Wegner. *Introducing psychology*. Macmillan, 2009.

[15] Eric L Schwartz. Spatial mapping in the primate sensory projection: analytic structure and relevance to perception. *Biological cybernetics*, 25(4):181–194, 1977.

[16] Eric L Schwartz. Computational anatomy and functional architecture of striate cortex: a spatial mapping approach to perceptual coding. *Vision research*, 20(8):645–669, 1980.