Department of Computer Science

BSc (Hons) Computer Science (Artificial Intelligence)

Academic Year 2020 - 2021

Retinal image analysis using Convolutional Neural Networks

Kittisak Tipakornrojanakit

1818694

A report submitted in partial fulfilment of the requirements for the degree of Bachelor of Science

Brunel University London
Department of Computer Science
Uxbridge
Middlesex
UB8 3PH
United Kingdom
T: +44 1895 203397
F: +44 (0) 1895 251686

Abstract

In the medical field of ophthalmology, retina examination is one of the methods used to perform diagnosis of the condition of the patient. This is because human eyes can reflect the general state of health of the entire human body. Many important eye diseases such as Diabetic Retinopathy (DR), Age-Related Macular Degeneration, and Glaucoma etc. as well as systemic diseases such as Diabetes manifest themselves in the retina. This problem gives rise to retinal imaging and image analysis method for ophthalmologists to perform an early detection of these risks. However, this method requires the ophthalmologists to manually diagnose the retinal images. The manual diagnosis of retinal images is a particularly time-consuming task that usually takes 7-14 days for a complete diagnosis, and albeit it is a computer-aided diagnosis, this method is still prone to human-error in the process.

In this project, the retinal image analysis problem is separated into 2 stages, which are vessel segmentation, and optic disc segmentation. In this approach, I have developed deep learning algorithms, particularly, Convolutional Neural Networks (CNNs) to perform segmentation to determine the eye structures and the lesion presents in the retina. A special architecture used in this study is a U-Net architecture, integrated with attention gates and replaced dropout with spatial dropout.

Throughout the course of this project, extensive training and evaluation were experimented, with the final model be able to achieved satisfying result on DRIVE and DRIONS-DB dataset

Acknowledgements

First and foremost, I would like to express my deep and sincere gratitude to my mother for her love, prayers, caring, and sacrifices for educating and preparing me for my future. The countless times you helped me throughout my journey. Your encouragement when the times got rough are much appreciated. It was a great comfort and relief to know that you were willing to provide everything you can give even though I know that you struggled a lot.

I would like to express my sincere gratitude to my supervisor Dr. Yongmin Li for the continuous support of my project, for his patience, motivation, enthusiasm, and immense knowledge. His guidance helped me in all the time of research and writing of this dissertation. I am truly grateful for this and could not have imaged having a better supervisor and mentor for my final year project.

I would also like to express my special gratitude and thanks to my second adviser, Dr. Jeff Wen for imparting his knowledge and expertise in this project.

Last but not least, I thank all my friends for helping me through the journey.

I certify that the work presented in the dissertation is my own unless referenced.

Signature	Kittisak Tipakornrojanakit
Date	26/03/2021

Total Words: 14358

Table of Contents

List of Tables

List of Figures

1 Introduction

This section introduces and explores the problems I plan to solve throughout the course of this project. Within this section of the report I will also clearly define an overall project aim, as well as produce a set of relevant objectives which will help me achieve my aim.

1.1 Problem Statement

Many retinal diseases has been a big threat to the retina that can ultimately result in blindness. To prevent the vision impairment caused by these diseases, patients under high risk of this had to be undertake regular eye examination [1]. However, the procedure of diagnosing and investigating as well as regular monitoring of the condition, requires the involvement of high attention from the ophthalmologists. This amount of workload is added by the number of patients constantly increasing worldwide, and with the shortages of physicians, this pressing issue will eventually exceed the current healthcare capabilities. In addition to the issue, the ophthalmologists are supposed to diagnose fundus images for several patients per day, this will resulting in them becoming easily fatigued, which can result in an inaccurate examination of the condition. Furthermore, despite firm guidelines in diagnosis, the diagnosis of certain images can vary significantly amongst different ophthalmologists as because human are subjective [2].

Presently, automatic image analysis algorithms based on image processing have been successful in various medical applications. Through automation of the retinal image analysis, additional patients can be effortlessly screened, providing the ophthalmologists to have more time for patients who require attention. Many researchers in this field have proposed and reported various analysis techniques for retinal images, with a number of the techniques has gain a noticeable improvement in performance. Nevertheless, several challenges and issues are still presents in various aspect of retinal image analysis such as localization, segmentation, classification. In this project, some of these issues are taken into consideration including Automation, Performance, Robustness and Generalization.

1.1.1 Automation

Significant amount of existing automated retinal image analysis models and techniques have been based on extracting the features from images manually through image processing [3] [4] [5] [6]. This causes the systems to depend on the type of features extracted from a specific dataset. Moreover, although the mentioned features might be applicable to some datasets, however, it also might not generalize well on some other datasets. Therefore, the issue that s hould be resolved is the automated retinal image analysis that should be independent of any human interaction.

1.1.2 Performance

Image processing in general, often require complex and high computations. This is due to many factors in the processing. Meanwhile, there are some methodologies that are able to obtain a relatively good performance, however, the time cost of their image processing are significantly a lot [3] [7] [8]. Furthermore, systems in clinical applications in the medical field are required to be accurate and reliable as they are important in the physicians' judgement in making decisions. Therefore, the critical need for time-efficient and cost-effective algorithms should be addressed as well as highly sensitive and accurate methods should be developed and provided.

1.1.3 Robustness and Generalization

In the field of ophthalmology, retinal images commonly contain noise due to interference. Sometimes the image's edges are blurred, boundaries not displaying correctly, and the brightness intensities are inconsistent. These problematic characteristics have significantly impact the process in analysing the image, which were addressed in many researches, especially in image segmentation tasks [9] [10]. Therefore, this issue should be addressed by developing robust image processing methods that can maintain its overall quality of analysis even on poor-quality images. In addition to this, the method should be able to generalize well when perform on various datasets.

For the past 5 years, deep learning algorithms have become popular in various medical field, particularly, medical image analysis [11]. Deep learning algorithms such as Convolutional Neural Networks (CNNs) plays a significant part in developing an automated feature learning algorithms.

1.2 Aims and Objectives

The aim of this project is to develop and implement automated segmentation using deep learning algorithms, particularly Convolutional Neural Networks (CNNs) for retinal image segmentation. This project will focus on the segmentation of retinal blood vessels and optic disc in order to firstly, determine the ocular structures, and secondly, allows for easier diagnosis. The goal of this project is to make segmentation and investigation of the retina more convenient for the ophthalmologist and eye care specialists for retinal diseases detection in early stages and treatment evaluation.

To achieve this aim, I will need to fulfil the following set of objectives:

1. Explore existing retinal image segmentation techniques and implementations by undertaking relevant research, and also through the analysis of applicable background literature.
2. Based on the research carried out through objective 1, identify which (if any) methods could be adopted and further developed to provide an effective, reliable solution for the problem of retinal image segmentation.
3. Before moving on to design and development of automated retinal image segmentation, the acquisition of labelled datasets consisting of marked structures of blood vessels and optic disc.
4. After identifying a feasible solution through objective 2, design and develop CNN model based on the models and techniques adapted from the feasible solution.
5. After planning and designing the system architecture and CNN model in objective 4, develop and implement the data pipeline and ultimately implement the automated image processing and segmentation.
6. Train the model through supervised learning, leveraging the datasets provided corresponding to each stage of segmentation tasks.
7. Conduct testing of the model on the test data. Important statistics and graphs such as accuracy or Dice score, will be produced to determine and validate the performance of the model in the model evaluation. After thoroughly testing the model, the result produced in object 7 will be evaluated to determine the performance of the CNN model.
8. Assessment of the overall project execution, and the description of future work should the project continues.

1.3 Project Approach

Investigation Phase

The beginning of the project, an investigation is a necessary step in the development of the project. In the first major part of this phase, I will explore the anatomy of the human eye and the retina assessment of the fundus image. Understanding concepts such as anatomy of the retina, fundus photography and the diagnosis of the retina, is essential to accurately develop an artificial intelligence-based method for segmentation. In the second major part, explore existing deep learning algorithms used in retinal image segmentation, by conducting extensive background literature review. Analysing the existing solution should then guide me to understanding the use of deep learning algorithms in this problem, where I will have the opportunity to consider what kind of approach could be adapted and used to implement an automated retinal image segmentation.

Methodology Phase

After investigating all the existing technologies, and approaches to similar problems, I will then proceed to establish the methodology in which the way the research will be carried out. Each step for developing a solution followed by the model evaluation plan will be laid out. As well as a research methodology, a software development will also be established. Together these will provide a structured way as to how to conduct the project and provide a natural flow of processes in which to follow during conduction.

Design and Build Phase

This major phase of the project, which predominantly involves design and development processes. Having established a methodology to approach for solution in this project, the iteration of design, build, and assessing model will be in this phase. The core architecture of the machine learning models and techniques will be presented in details, in conjunction with the core architecture of the system with considerations on the best practises.

Implementation Phase

In this main phase of the overall project, this phase will put together the knowledge gained in the methodology phase and the designs and building of the models produced in design and build phase to implement the software solution.

Evaluation Phase

In this last phase, the models developed as well as software will be evaluated, producing all the important graphs to validate the performance of the model used in the solution.

1.4 Dissertation Outline

The following flowchart diagram illustrates the structure of my dissertation. Chapters are represented by colour-coordinated boxes, depending on the project phase they fall under. Labels also represent which objectives are completed within each chapter.

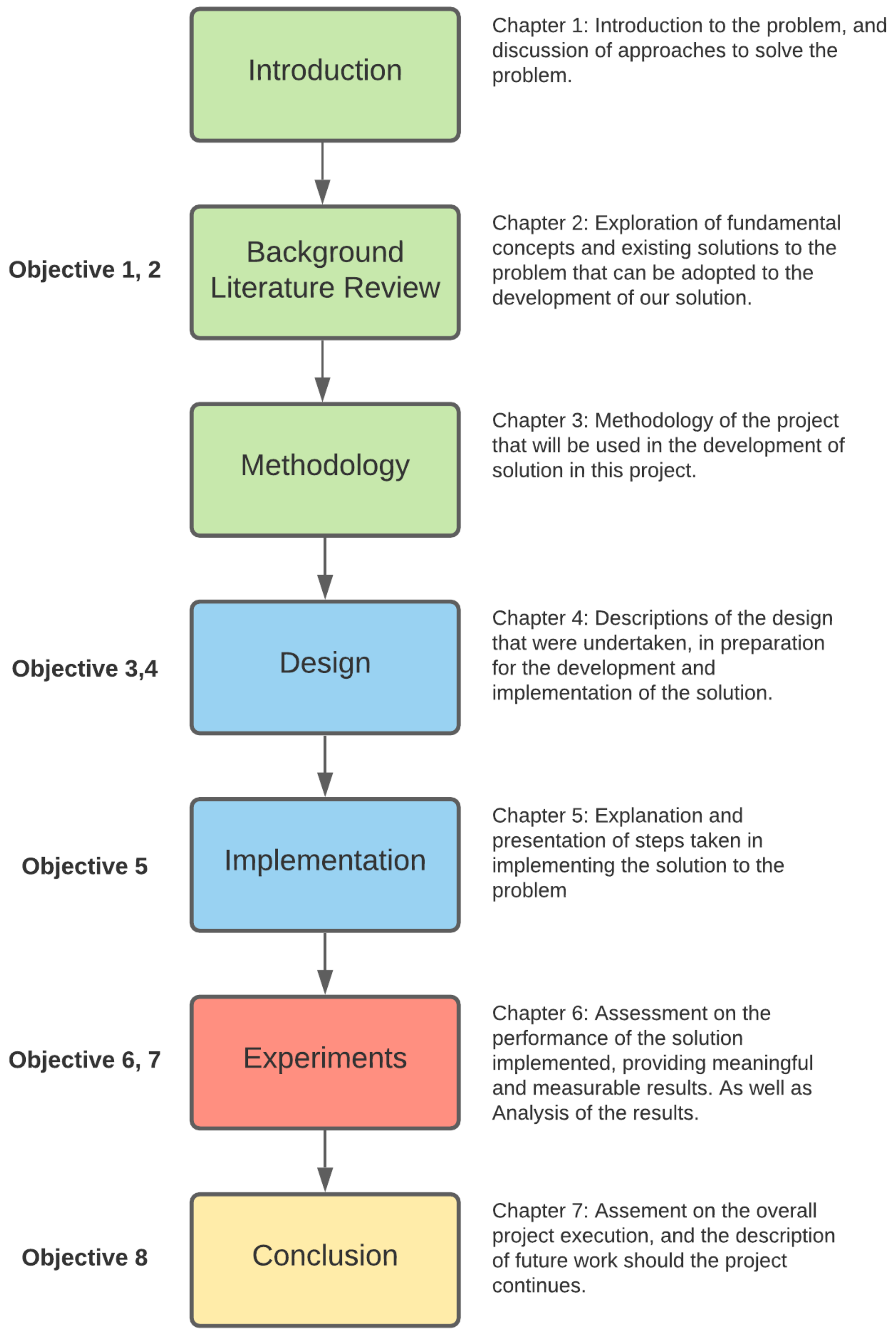

Introduction
Chapter 1: Introduction to the problem, and discussion of approaches to solve the problem.
Objective 1, 2
Background Literature Review
Chapter 2: Exploration of fundamental concepts and existing solutions to the problem that can be adopted to the development of our solution.
Methodology
Chapter 3: Methodology of the project that will be used in the development of solution in this project.
Objective 3,4
Design
Chapter 4: Descriptions of the design that were undertaken, in preparation for the development and implementation of the solution.
Objective 5
Implementation
Chapter 5: Explanation and presentation of steps taken in implementing the solution to the problem
Objective 6, 7
Experiments
Chapter 6: Assessment on the performance of the solution implemented, providing meaningful and measurable results. As well as Analysis of the results.
Objective 8
Conclusion
Chapter 7: Assement on the overall project execution, and the description of future work should the project continues.

Figure 1. Flow diagram illustrating the structure of the dissertation.

2 Background

This section explores the problem in greater detail through reviewing literature reviews and other research materials. Within this section of the report I will also investigate some of the existing solutions and approaches to the problem.

2.1 Inside The Human Eye

If we were to look through the fundus camera, we would see the retina in which the three components that characterise it are the macula, the optic nerve circle, and the blood vessels branched in the retina. The macula is the centre darkish spot in the retina where the eye has the ability to best recognise the visual details. The damage of the macula will cause loss of the central vision. The optic nerve is the passage for the million of nerves that are linked to the brain. The blood vessels, just like any part of human body, provides nutrients and oxygen to the cells residing in the retina [13] [14].

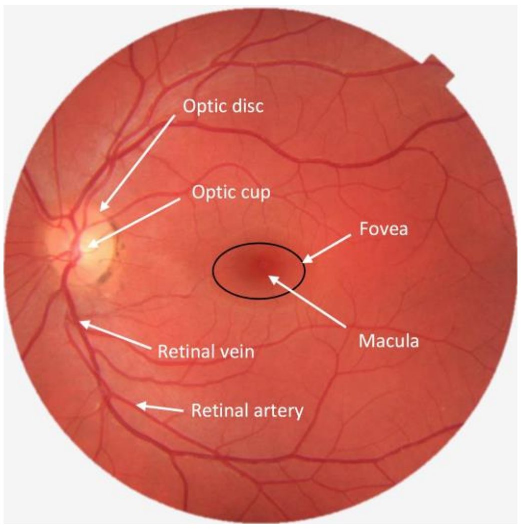


Figure 2. An example of fundus image illustrating the key retinal anatomical structures.

2.2 Retinal Diseases

Retinal diseases are the diseases of the eye that affect the vital tissue and cause various changes in the vision. One of the severe changes is vision impairment, causing blindness. Within the category of diseases, Diabetic Retinopathy and Glaucoma will be discussed below.

<u>Diabetic Retinopathy</u>

Diabetic Retinopathy (DR) is a retinal diseases associated with diabetes. Diabetes occurs when the glucose levels in the blood is abnormally high and does not stay at a safe range. The high glucose levels in the blood can also causes damage to the vessels supplying blood to the retina such as haemorrhages (bleeding). Subsequently, results in distorted vision [15].

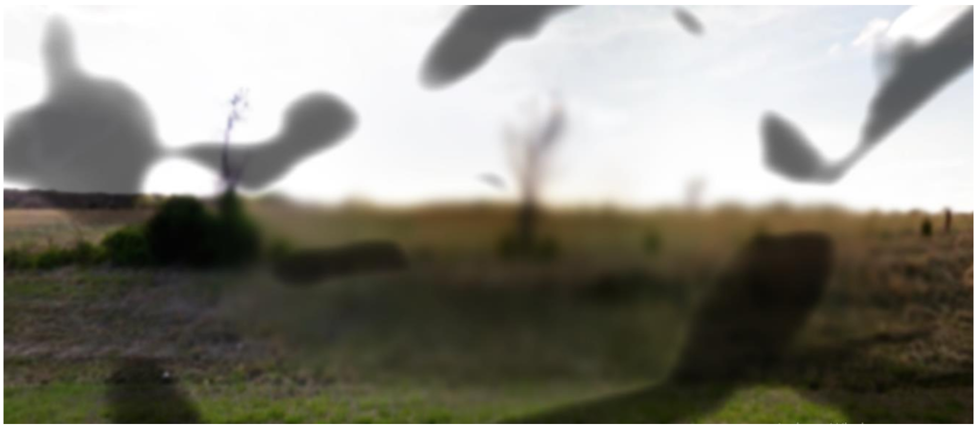


Figure 3. Example of how diabetic retinopathy affect a person vision

DR is categorized into 2 class: non-proliferative DR (NPDR) and proliferative DR (PDR). In PDR stage, new abnormal blood vessels appear on the surface of the retina, which can lead to scarring and cell loss in the retina [16]. It is the main reason of poor vision in people with type 1 diabetes or type 2 diabetes. Figure 5 illustrates the damage in the retina caused by DR.

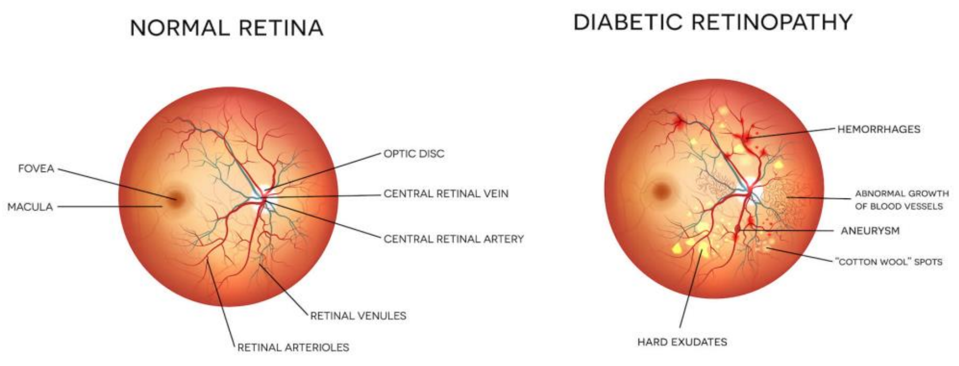


Figure 4. Illustration of a normal retina and a retina with diabetic retinopathy

<u>Glaucoma</u>

Glaucoma is one of the common retinal disease where the optic nerve becomes damaged. The common causes for this is the build-up of pressure in the eye when eye fluid are not drained properly and it builds up in the front part of the eye.

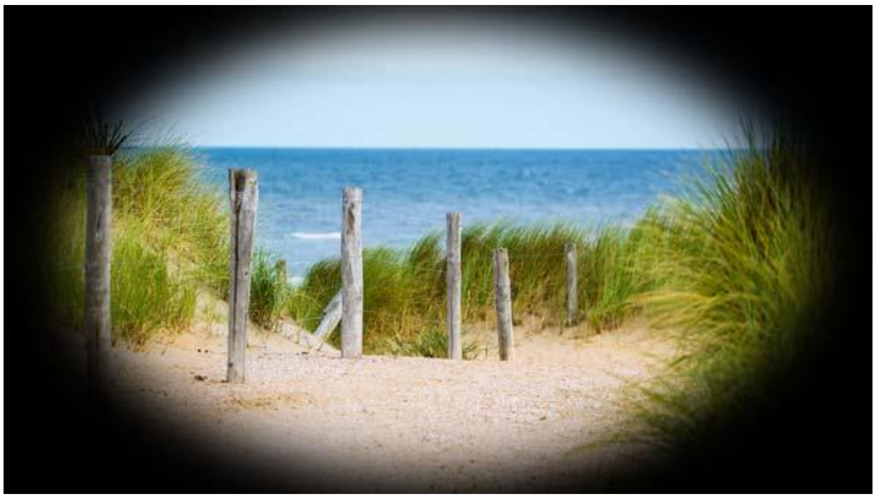


Figure 5. An example of how glaucoma affect a person's vision

In retinal images, the optic nerve can be seen, in which its conditions can be diagnose by the ophthalmologists. An indication of glaucoma can be checked by looking at the ratio between the optic cup and the optic nerve head [17].

The two retinal diseases mentioned above are very common in recent times. Thus, early detection and treatment of these retinal conditions will help in preventing the progression of the disease. Consequently, this leads us to the next background study.

2.3 Retinal Imaging Modalities – Fundus Photography

In medical field of ophthalmology, fundus photography is one of the most common way to capture an image of the retina in order to enable the ophthalmologists to diagnose the eye.

<u>Fundus Photography</u>

Fundus Photography is the earliest type of retinal imaging techniques used in capturing serial photographs of the interior of the eye through the pupil. The captured image is called fundus image, or retinal image, in which it captures the structures of the retina in full colour [18]. This is called colour fundus photography. The following figures shows the fundus camera, and how the retinal image are captured:

Figure 6. An example of a fundus camera: Topcon TRC-NW8F Plus Retinal Camera

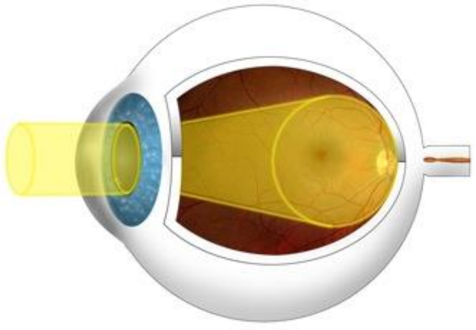


Figure 7. Illustration of how the retina image is captured

Fundus photography is important as it allows the documentation of the retinal diseases as well as records its progress in patients' eye. Figure 9 below shows stages of DR in fundus images:

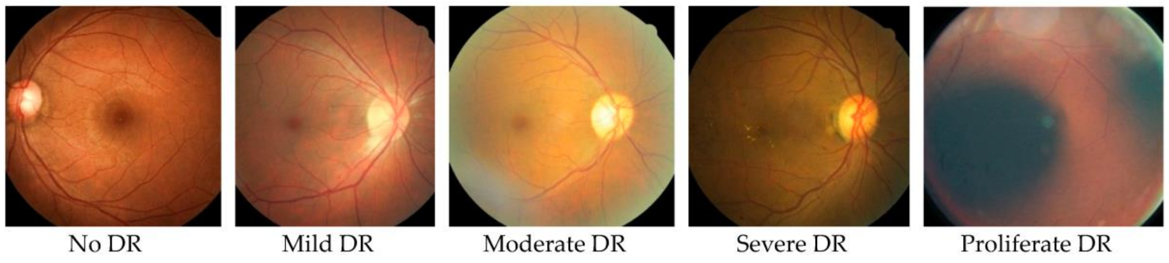


Figure 8. Progression of diabetic retinopathy can be examined in fundus images.

2.4 Medical Image Analysis

Recently, the development of automatic analysis of retinal diseases from digital retinal images has received increasing attention in the medical image processing community. This gives rise to the continuous development of computer-aided diagnosis systems for assisting healthcare workers in diagnosing the retinal conditions. Many techniques of digital image processing have been developed in these systems to extract important features of the retinal images. There are many sophisticated algorithms that can be used for different purposes in image processing. In this section, low-level image processing and high-level image processing are described and its purposes are highlighted. Some techniques from both level will be taken into consideration during the experimentations.

2.4.1 Low-level Image Processing

Low-level image processing algorithms perform simple processing on the image in order to process it further by high-level image analysis algorithms [19].

Image Enhancement

Image enhancement is one of the most common techniques used in many applications in order to improve the visual appearance of the images. There are many ways to enhance an image such as by sharpening the boundaries or increasing/decreasing the contrast scale of the image. An example of original image and contrast-enhanced image by contrast-limited adaptive histogram equalization (CLAHE) [20] [21] algorithm is shown in Figure 10 below.

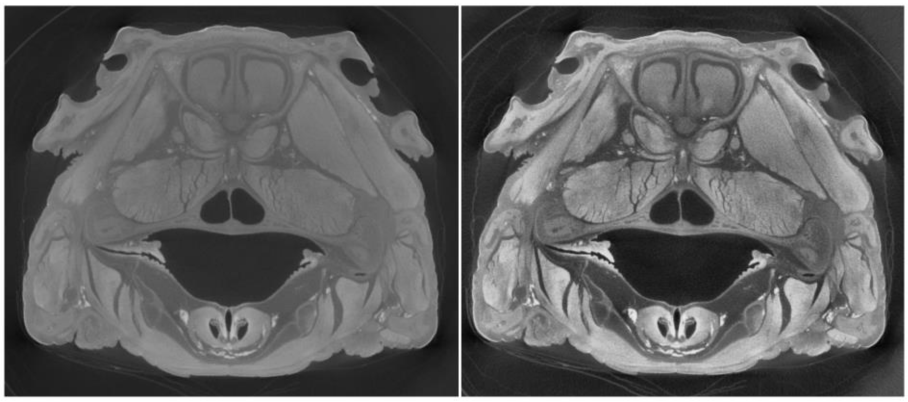


Figure 9. Images from DiceCT, showing the head of a two-year old Alligator head in transverse views. Original image is on the left, and the image filtered using CLAHE is on the right.

CLAHE and other different enhancement techniques [20] have been considered in this project in order to improve the visual appearance of the retinal images.

2.4.2 High-level Image Processing

High-level image processing involves in higher level of interpretation of the images. Applications of high-level image processing in retinal images are such as classifying and grading the severity of a disease by analysing the disease's characteristics, or locates the region of interest (ROI) by segmenting the desired object in an image. Since this project involves in segmenting the retinal image, object segmentation are taken into consideration.

Image Segmentation

Image segmentation is a high-level image process by partitioning an image into multiple segments. The segments comprises pixels that have common characteristics such as colours, or texture which are assigned a label so that the pixels with same characteristics have the same label. Significant research effort are put into developing segmentation techniques in solving issues in many areas such as medical imaging. Image segmentation algorithms can be separated into five main methods: thresholding [22], edge-based [23], region-based [24], model-based [25], and machine learning-based [26]. Since this project is primarily focus on developing an automatic image processing using artificial intelligence, machine learning-based method will be adapted into this project.

2.5 Artificial Intelligence, Machine Learning, and Deep Learning

Artificial Intelligence (AI) refers to the technologies that simulates the human intelligence in machine. In recent times, there are many successful applications of an AI-like behaviour such as self-driving cars, smart homes, personal assistants, and many other emerging technologies. Machine Learning (ML) is a subset in the field of AI in which currently has been widely developed in various implementations. A 'deeper' subset in the field of AI is the new emerging trend called Deep Learning (DL). The term 'deep learning' come from a newly strategized implementation to generate deep hierarchies of non-linear features whilst overcoming the vanishing gradients problem. Vanishing gradients is a common issue which occurs when the gradients in the deep layers become too small to provide a learning signal and get stuck in apparent local minima. This issue are commonly mitigated in the developed deep learning architectures, allowing the layers to be trained effectively. Figure 11 shows a Venn diagram illustrating the set AI and its subsets.

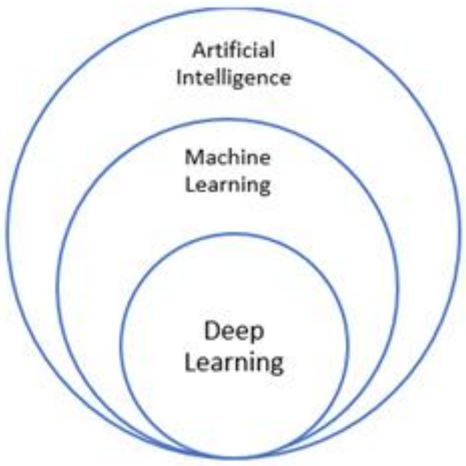


Figure 10. Venn diagram illustrating the subsets of AI

2.5.1 Existing Deep Learning Architectures

Many existing developed deep learning architectures have been introduced such as deep belief networks (DBNs) [27], deep Boltzmann machines (DBMs) [28], generative adversarial networks (GAN) [29], Long short-term memory recurrent neural network architectures (LSTM) [30] as unsupervised generative models, and architectures such as deep convolutional neural networks (CNNs) [31] as supervised discriminative models. These architectures have been widely and successfully applied to many applications such as automatic speech recognition [32], achieving state-of-the-art results. In this project, CNNs architecture has been adopted as the feature learning algorithms because of its solid ability in detecting the desired features in the images.

2.6 Existing Deep CNNs Applications in Retinal Image Analysis

One of the well-known CNN architecture in the field of medical image segmentation is an architecture called U-Net [33]. U-Net was proposed in Ronneberger et al. (2015) paper. The reason it is called U-Net is because of its U-shaped architecture, consisting of a contracting path to capture context and a symmetric expanding path that enables precise localization. The performance of U-Net does not go unnoticed. Many recent methods for both retinal vessel segmentation and optic disc segmentation are based on U-Net.

For example, Wang et al. (2019) [34] refined U-Net into the Dual Encoding U-Net (DEU-Net) and reported an enhanced network better capable in segmenting the retinal vessels in an end-to-end and pixel-to-pixel way. Guo et al. (2020) [35] proposed Spatial Attention U-Net (SA-UNet) which introduces a spatial attention module (SAM) which helps the network to focus on important features and suppress unncessary ones to improve the network’s representation capability, resulting in state-of-the-art performance. In addition to vessel segmentation, Zhang et al. (2019) [36] proposed Attention Guided Network (AG-Net) which introduces an attention mechanism called “Attention Guide Filter” to better preserve the structural information, and reported a more effective segmentation of the optic disc.

2.7 Summary

This chapter has presented an overview of the anatomical structure of the eye, retinal diseases, retinal imaging modalities, and the existing methods in retinal image segmentation. From the works reported in this chapter, it can be noticed that the U-Net have demonstrated a state-of-the-art performance in many applications on variety types of medical images. As a result of that, U-Net will be adapted to solve the problems presented in this project.

3 Methodology

3.1 Introduction

In this chapter, the methodology to provide the solution will be described and explained. This project are partitioned into 2 sections: retinal vessel segmentation, and retinal optic disc segmentation. The details on how the solution will be defined will be fully explained as well. Lastly, the evaluation of the model will also be described in model evaluation section as well as how the completed automated retinal image segmentation will be evaluated to meet the aim of this project.

3.2 Research Approach

In this section, the approach to segmenting the retinal vessels and optic disc will lead from Chapter 2 Background, taking inspiration from multiple sources and apply and adapt them to this problem. The overall aim of this project is to address the issue of manual segmentation by providing a system solution.

3.2.1 Data Gathering

Data gathering is an essential task to any kind of research. The following table shows the datasets that will be used in the segmentation tasks.

Table 1. The datasets that will be used in segmentation tasks.

Datasets	**DRIVE**	**DRIONS-DB**
Source	Diabetic Retinopathy Screening Program, Netherlands	Ophthalmology Service, Spain
Total number	40	110
Train / Test Number	20 / 20	90 / 19
Resolution $W \times H$ (pixels)	565×584	600×400
Resized $W \times H$ (pixels)	512×512	512×512

3.2.2 Data Augmentation

Data augmentation is a necessary technique in data analysis for tackling small number of data available in which it is used to increase the amount of data by adding slightly modified copies of already existing data. As we can see, for example, DRIVE dataset have 40 images available, and only 20 images is being used for training, data augmentation will be used in order to increase the training data size and also effectively helps reduce over-fitting when training a CNN model.

3.2.3 CNN Model – U-Net

As described in the 2.7 Summary, U-Net will be used in segmenting vessels and the optic disc. Although the original U-Net achieved a state-of-the-art results, there are many proposed mechanisms that were proven to enhance the model in segmentation tasks such as Attention Gates (AGs) [37], Spatial Dropout [38], and many more others. Therefore, in the experimentation, these mechanisms will be implemented and experimented on.

3.3 Model Evaluation

Evaluation of a model is a mandatory step in machine learning. It is important that the model performance is measured by comparing the actual ground-truth that the model should output to the predicted output. In binary segmentation, some metrics are more effective in evaluating the model.

3.3.1 Metrics

Two different metrics will be used to evaluate the performance of the model: Accuracy and Dice Coefficient.

1. **Area under the ROC (AUROC/AUC):** the area under the receiver operating characteristic is a performance metric that is commonly used in machine learning community to evaluate the model for its 'discrimination' ability. For binary classification/segmentation model, if the AUC value is 1, this means that the predicted output of the segmentation map is a perfect segmentation [39].

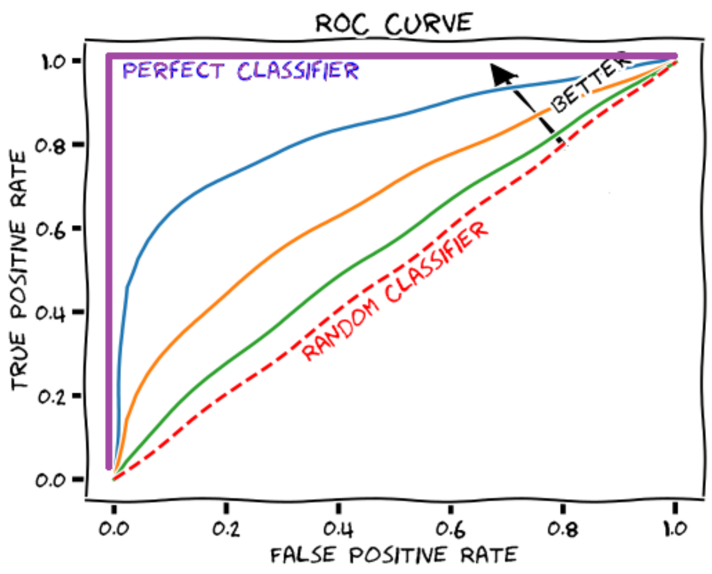


Figure 11. Different ROC Curves

2. **Dice Coefficient:** the dice coefficient is the measure of overlapping similarity between the predicted output values (P) and the actual ground-truth values (G). This metric is commonly used to evaluate segmentation performance [40].

3.3.2 Validation and Evaluation

For every task in the project, the dataset corresponding to the task will be split using the train-test-validation split technique. This standard technique allows for the model to cover all data-point belonging to a single dataset. The model will be trained using the training set and validate using the validation set and then finally evaluate the model on the test data. Figure 13 illustrates a full dataset being split into training, validation, and testing set.

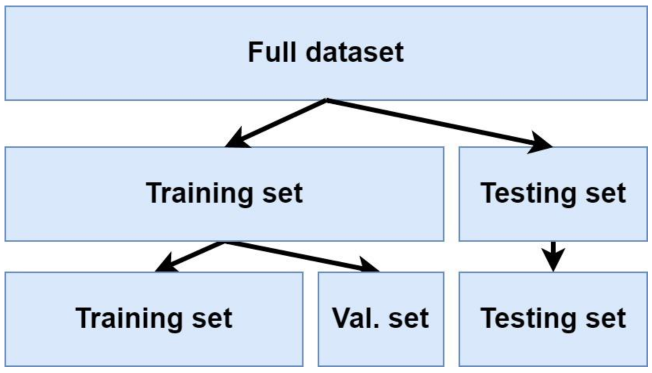


Figure 12. Illustration of a full dataset splitting into train-validate-test sets

As because we have already split the dataset into 3 parts, holding out the testing set, we can now begin evaluation of the model by using the testing set. This will allow us to see how the model perform on the 'real world' data (unseen data).

3.4 Summary

In this chapter, we defined the key components of this project. The methodological approach were also explained in detail in this chapter. The approach was broken down into smaller sub-steps: data preparation, approach on designing the model, as well as model testing and evaluation. In short, this chapter provides details on how the project will proceed. Therefore, a planned and structured methodology approach should lead to a solution that can achieve the aim of the project. This chapter further fulfils the objective 2.

4 Design

4.1 Introduction

In this chapter, the documentation of the design were shown. This is to prepare us for the development of the solution. Within this chapter, system architecture, programming language, and framework are also identified for the implementation of the system.

4.2 System Design

The key requirements in producing a good system are reproducibility, generalizability, and scalability. A good and reproducible pipeline were proposed in Sugimura et al. (2018) [41] in which they states that in the designing of the system architecture, the concepts should be treated as a first-class citizen [42]. Also, in order for the same system architecture to achieve generalizability and scalability, the project should be split into components. Ultimately, having a good structure allows for easy understanding of the system and will significantly help in developing automated image segmentation system.

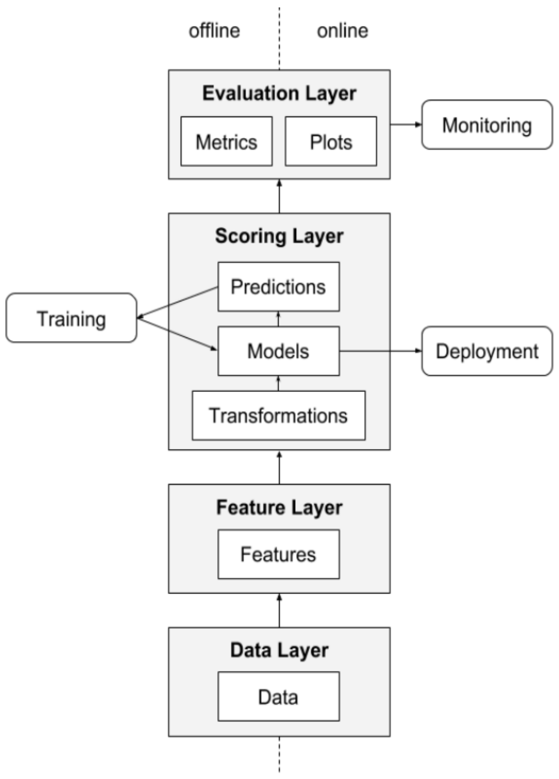
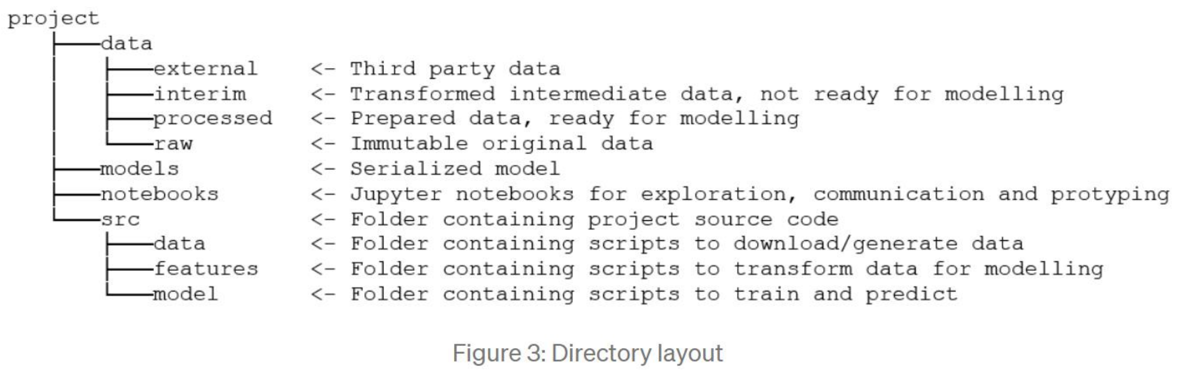


Figure 13. Sugimura, P.Hartl, F. 2018 proposed project components layout

The above layout is only the layout of the components of the data pipeline, so the structure of the project is still needed to be designed. The proposed structure layout from Ericson et al. (2020). This is formulated from Microsoft – Team Data Science Process (TDSP).



```
project
    ├──data
    │   ├──external    <- Third party data
    │   ├──interim     <- Transformed intermediate data, not ready for modelling
    │   ├──processed   <- Prepared data, ready for modelling
    │   └──raw         <- Immutable original data
    ├──models          <- Serialized model
    ├──notebooks       <- Jupyter notebooks for exploration, communication and protyping
    └──src             <- Folder containing project source code
        ├──data        <- Folder containing scripts to download/generate data
        ├──features    <- Folder containing scripts to transform data for modelling
        └──model       <- Folder containing scripts to train and predict
```

Figure 3: Directory layout

Figure 14. Directory Layout proposed in Ericson et al. 2020

The project structure used in this project are not exactly the same as the above structure, only taking them as inspiration. Before moving on the next sub-section, the project are all implemented for offline use for simplicity.

4.2.1 Data Loading

The data folder groups all the datasets into one folder, in order for simplicity in accessing the data. In structuring the data folder this way, this allows the access to the data inside datasets much easier as it is all gathered up inside one folder. The following figure shows a simple structure inside the data folder with datasets as sub-directories:

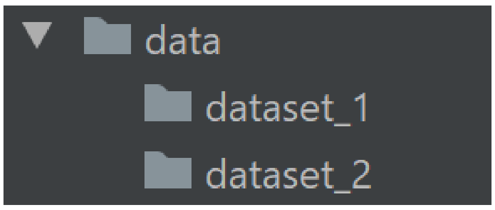


Figure 15. Illustratation of simple data folder.

A script file is responsible for accessing these data inside the specified datasets, in which it reads the data into memory, and transform the data into readable form, and sends it through the functions responsible for preparing the dataset. This led us to the next sub-section: Data preparation.

4.2.2 Data preparation

Typically, in computer science, the use of computer memory is one of the major factor in this field. In this project, much of datasets used in this project consists of many large medical images which can take a lot of space in the random access memory (RAM) which can have an impact in the processing time of the computer. This also affect the time to read raw images in order to prepare the dataset for further analysis. The solution to this problem, is the use of file formatting, and that is Hierarchical Data Format (HDF). HDF is a set of file formats, most recent is HDF5, designed to store and organize arbitrary amount of data [43].

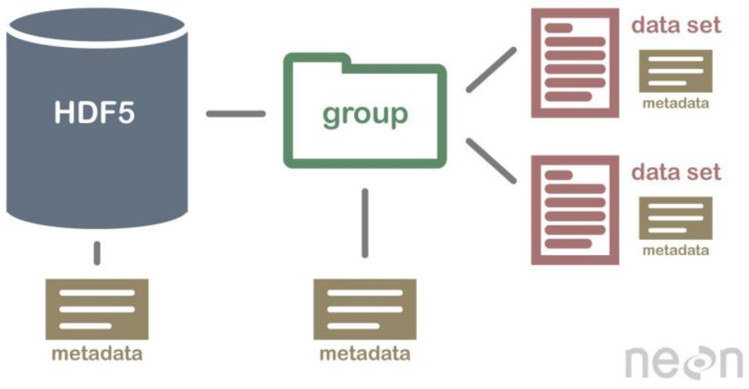


Figure 16. Illustration of a HDF5 file with datasets metadata inside.

Each HDF5 files contain metadata of images which are separated into train, validate, and test files. Finally, the HDF5 files can be read whenever we want in order to perform the next step: Data processing.

4.2.3 Data pre-processing

In this final stage of the data pipeline, the data are read into the system and be processed in order to transform it into the special form for model training and validating. This step in the pipeline can also be called pre-processing step, and it is important in machine learning. Data pre-processing is crucial in machine learning as the quality of the data and the desired features directly affects the performance of our model. Therefore, pre-processing the data will be integrated into the data pipeline. There are various way to pre-process the data, for example, standardization (normalization) is one of the common procedure used in pre-processing step. As the data goes through each stage, the current data in the memory are now the transformed and processed data, ready for feeding it into the model in order to begin feature learning so that the model can output the correct prediction we want it, and ultimately achieve the goal of this project.

4.3 Software Implementation

4.3.1 Programming Language and Platform

Many machine learning engineers and scientists uses varieties of programming language in the field of data science. For this project, Python was chosen as the programming language, and Tensorflow with Keras was chosen as the framework for implementing machine learning algorithms and model training/testing.

Python: Since we are working with machine learning, Python is a good choice as a programming language. This is because Python is dynamically-typed and garbage-collected. It is also a multi-paradigm programming language as stated in its design philosophy. This means that object-oriented programming (OOP) and structured programming as well as functional programming are fully supported. In addition to the definition, the best aspect of Python is its widely available libraries in which many covers the data science aspect. Thus, Python is a perfect fit to implement the solution of this project.

Tensorflow: Tensorflow with Keras is a good framework, especially the latest version, Tensorflow 2.0. Tensorflow was chosen because of its easy-to-understand and flexible architecture which is good for beginners going into data science and machine learning. Furthermore, the newest version has integrated with Keras, thus there are more options when it comes to training and inference of deep neural networks.

4.3.2 Object-Oriented Programming (OOP)

Object-oriented programming is a programming paradigm that relies on the concept of classes and objects [44]. It is used to structure a system into simple, reusable pieces of code blueprints (usually called classes), which are used to create individual instances of objects. This approach will be used in the development of the solution. Figure 19 illustrates the concept of OOP.

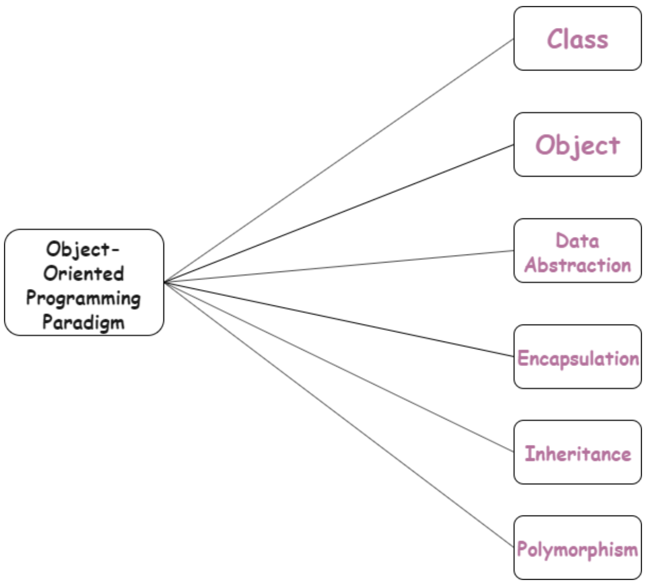


Figure 17. Concepts in OOP Paradigm

Although there is little to no advantage in using OOP when it comes to data science, the final solution should be in reproducible, scalable, and clean code that allow others to understand the structure and the flow of the program easily.

4.3.3 Python IDE

To achieve this project's aim, the solution has to be implemented in the chosen language which is Python. There are many available platform for implementing Python codes, for example, text editors such as Sublime text, and Vim, or full-fledged IDE such as Pydev, and Pycharm. As because implementation of the solution will be developed in an OOP approach, the challenge of writing a reproducibility, and scalability code is a significant task when using text editors. Thus, Pycharm was chosen as an IDE for an implementation of the solution in order to allow the solution be developed in OOP approach as well as making the code looks nicer.

4.3.4 Version Control System

Version Control Systems (VCSs) enables for effective tracking of the changes and the workflow. Github is one of many examples of VCSs. VCSs provides features such as branching which allows the uninterrupted workflow in the environment of CI/CD. There can be many branches in a single repository. This means that changes in branches can be developed without affecting the master branch and others. When a branch is ready to be integrated, it can then finally be merged back into the master branch.

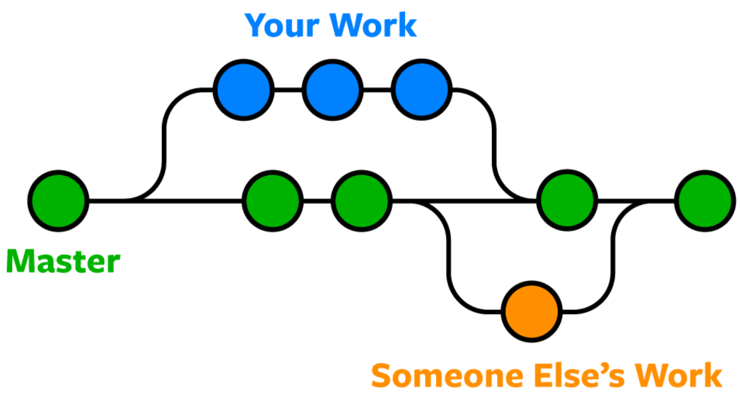


Figure 18. An example of a repository with branches.

4.4 Model Design

As mentioned in 2.7 Summary, U-Net will be adapted and improved upon for our segmentation tasks. The base architecture of our U-Net model are as followed: The contracting path consists of the repeated 3×3 convolutions, each one followed by a rectified linear unit (ReLU), arriving at the bottom with a 2×2 max pooling with a stride 2 for downsampling. The expansive path consists of upsampling followed by a 2×2 deconvolution, halving the number of feature channels, and a concatenation with the feature from skipping connections. Then followed by, two 3×3 convolutions, each one followed by a ReLU. Batch normalization are added after every convolutional layer followed by a dropout for regularization. The finaly layer is a 1×1 convolution applied to map each feature vector to the desired number of class. In segmentation tasks are binary segmentation thus the number of class is 1.

The architecture of the U-Net in Figure 20 is the original U-Net architecture presented in the (Ronneberger, O., Fischer, P., and Brox, T., 2015).

In addition, as mentioned in 2.7 Summary, there are some improvements that can make the U-Net be able to learn the feature much better and improve the overall accuracy and quality of the segmentations. The following will be implemented into U-Net in a way that can be swapped through the parameters:

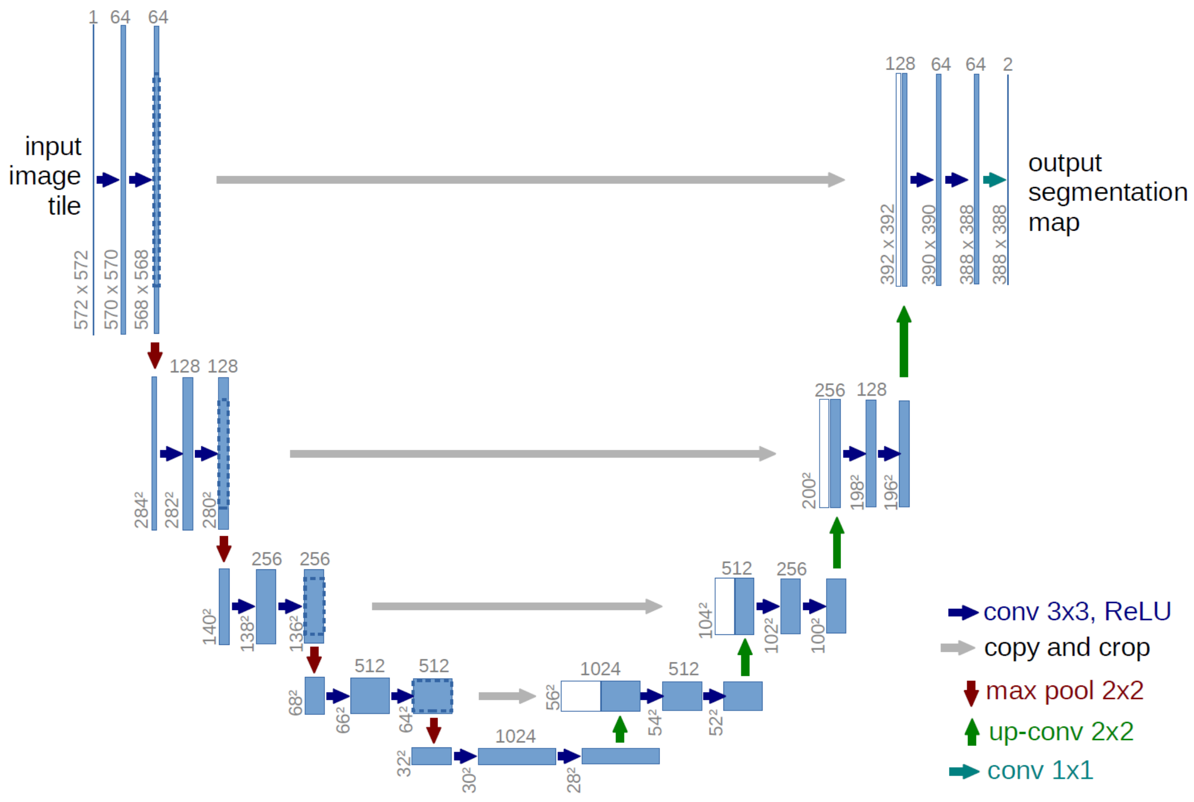


Figure 19. Original U-Net architecture

1. **SpatialDropout2D:** SpatialDropout is a dropout layer works differently by looking at the adjacent pixels in the dropped-out feature map and calculate it to be either all 0 (dropped-out) or all active. They found that this modified dropout implementation improves performance, especially in dataset where the training set size is small. Furthermore, SpatialDropout layer are implemented into Keras in 3 versions, which are 1D, 2D, and 3D. The 2D version were chosen because our feature maps are 2D.

2. **Attention Gates (AGs):** AGs were proposed in (Oktay et al. (2018) [37], where AGs are incorporated into the standard U-Net architecture to highlight salient features that are passed through skip connections. Information extracted from coarse scale is used in gating to disambiguate irrelevant and noisy responses in skip connections. This is performed right before the concatenation operation to merge only relevant activations. Thus, this method was chosen in order to experiment the model for better improvement on accuracy and quality of the segmentations.

4.5 Summary

In summary, the project's implementation will be implemented in Python using the Tensorflow framework on the Pycharm IDE The system will be structured using the inspiration from the proposed structure as defined in 4.2 System Design. This will make use of object-oriented paradigm that allows for reproducibility, generality, and scalability of the solution.

The design of the model that will be used in the experimentation have been demonstrated in 4.4 Model Design where the existing model – U-Net – will be adapted and implemented in the way that can be modified through the parameters.

By putting together what we defined in this chapter will allow us to achieve the aim of this project of automatic retinal image analysis by providing a fast segmentation of retinal vessels and optic disc, and therefore helps ophthalmologist and eye doctor in diagnosing of retinal images.

This chapter achieves – objectives by producing the architectural aspects of the solution. The design of the machine learning aspects of the solution also feed in this.

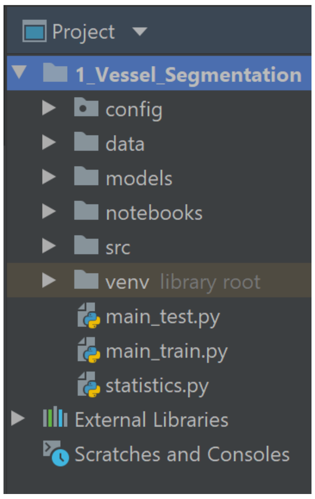
5 Implementation

5.1 Introduction

In this chapter, the implementation of the retinal image analysis parts will be provided and explained in details. The implementation is separated into 2 parts: vessel segmentation, optic disc segmentation.

5.2 System Architecture

Various system architecture were described in section 4.2 System Design. Figure below shows the layout of the project.

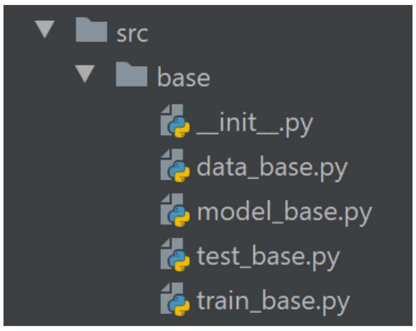


```
Project
▼ 1_Vessel_Segmentation
    ▶ config
    ▶ data
    ▶ models
    ▶ notebooks
    ▶ src
    ▶ venv  library root
      main_test.py
      main_train.py
      statistics.py
▶ External Libraries
  Scratches and Consoles
```

Figure 20. Project structure in vessel segmentation task.

5.2.1 Classes and Inheritance

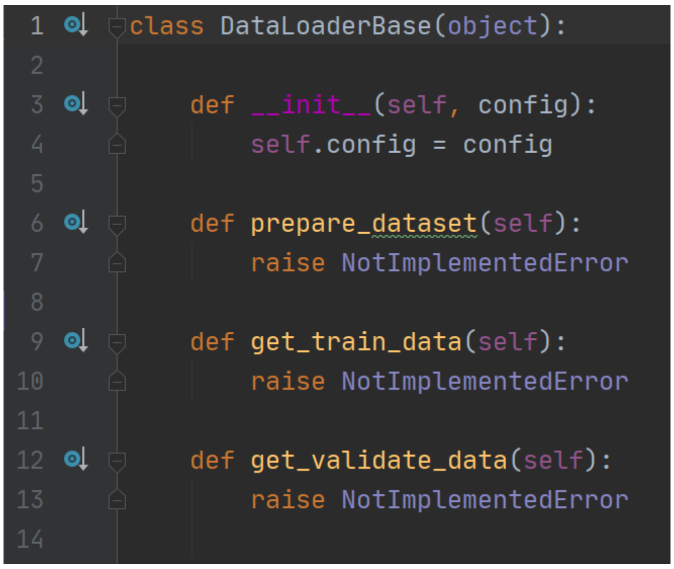
Since we are using OOP approach for making a robust software, the main classes were implemented through the use of inheritance. Inside the 'base' folder of the 'src' folder contains the parent classes for the main classes.



```
▼ src
    ▼ base
          __init__.py
          data_base.py
          model_base.py
          test_base.py
          train_base.py
```

Figure 21. Base classes

The base classes allows for the implementation of children classes that inherits its parent's properties. The following code snippet shows an example of a parent class with defined functions:



```python
class DataLoaderBase(object):

    def __init__(self, config):
        self.config = config

    def prepare_dataset(self):
        raise NotImplementedError

    def get_train_data(self):
        raise NotImplementedError

    def get_validate_data(self):
        raise NotImplementedError

```

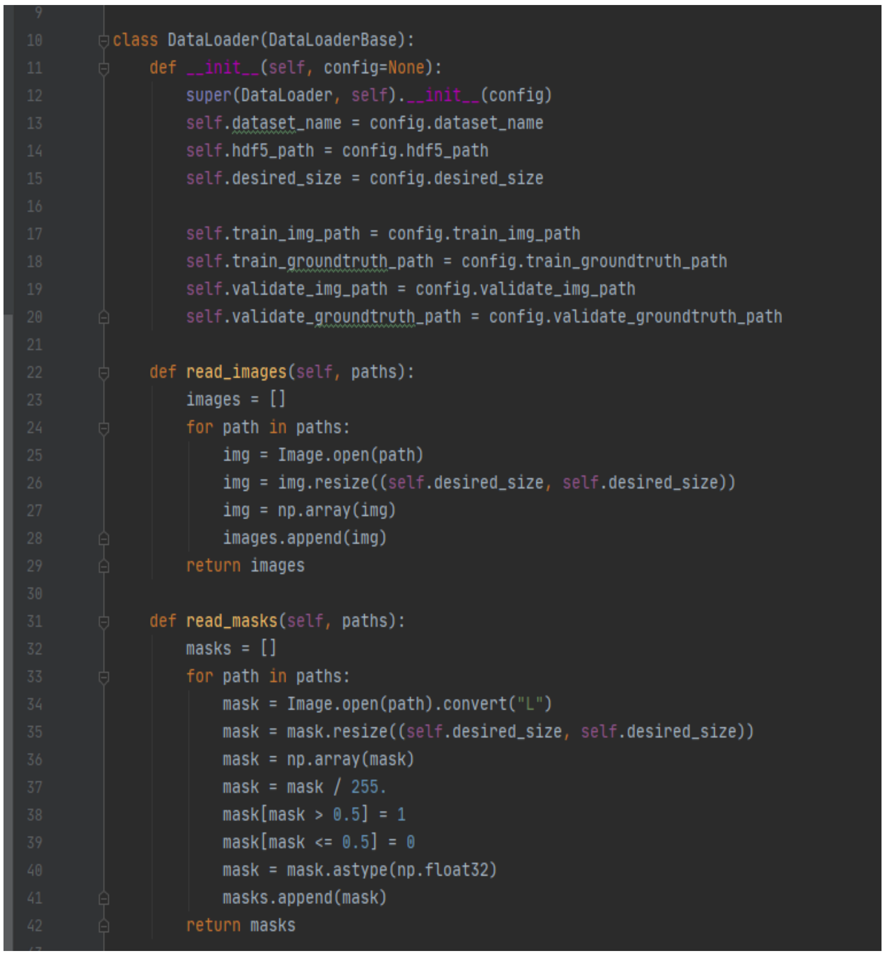
Figure 22. Code snippet 1 - an example of a base class - DataLoaderBase.py

The children classes consists of Dataloader, Model, Test, and Train classes. These four inherits the functions of its parent classes: data_base, model_base, test_base, train_base, respectively.

5.2.2 Data Loader and Preparation

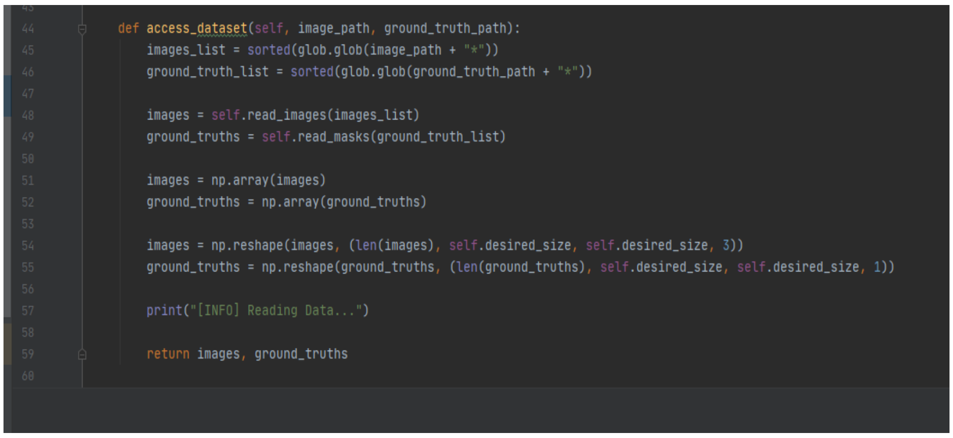
The data loder is implemented inside the data.py which provides a natural flow of data and its conversion to HDF5 file format in one go, having it ready in the form of standard numpy array and saving it into the HDF5 file, separating it based on the split of train and validate set. Both retinal images and its corresponding vessels ground-truth are read and converted into numpy form, keeping it in numpy array. This can be seen in Code snippet – below.

The code snippet 2 showing, functions are then put into the access_dataset function. This function takes in the path to both the images and groundtruths and then call the read functions which then store the images and ground-truths data in each array respectively. After that, both arrays will be converted into numpy array of type ndarray. Finally, the numpy arrays of images and ground-truths are returned by this function.



```python
class DataLoader(DataLoaderBase):
    def __init__(self, config=None):
        super(DataLoader, self).__init__(config)
        self.dataset_name = config.dataset_name
        self.hdf5_path = config.hdf5_path
        self.desired_size = config.desired_size

        self.train_img_path = config.train_img_path
        self.train_groundtruth_path = config.train_groundtruth_path
        self.validate_img_path = config.validate_img_path
        self.validate_groundtruth_path = config.validate_groundtruth_path

    def read_images(self, paths):
        images = []
        for path in paths:
            img = Image.open(path)
            img = img.resize((self.desired_size, self.desired_size))
            img = np.array(img)
            images.append(img)
        return images

    def read_masks(self, paths):
        masks = []
        for path in paths:
            mask = Image.open(path).convert("L")
            mask = mask.resize((self.desired_size, self.desired_size))
            mask = np.array(mask)
            mask = mask / 255.
            mask[mask > 0.5] = 1
            mask[mask <= 0.5] = 0
            mask = mask.astype(np.float32)
            masks.append(mask)
        return masks
```

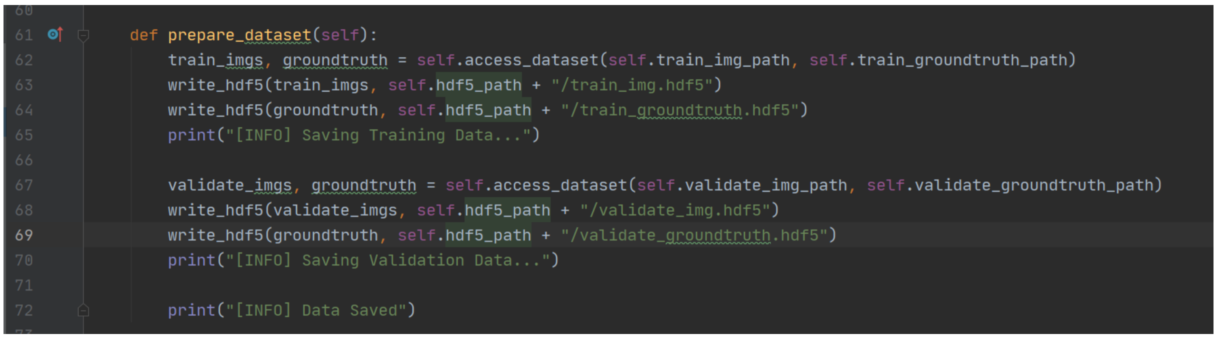
Figure 23. Code snippet 2 – DataLoader Class with functions for reading images and its ground-truths.



```python
    def access_dataset(self, image_path, ground_truth_path):
        images_list = sorted(glob.glob(image_path + "*"))
        ground_truth_list = sorted(glob.glob(ground_truth_path + "*"))

        images = self.read_images(images_list)
        ground_truths = self.read_masks(ground_truth_list)

        images = np.array(images)
        ground_truths = np.array(ground_truths)

        images = np.reshape(images, (len(images), self.desired_size, self.desired_size, 3))
        ground_truths = np.reshape(ground_truths, (len(ground_truths), self.desired_size, self.desired_size, 1))

        print("[INFO] Reading Data...")

        return images, ground_truths
```

Figure 24. Code snippet 3 - Function for accesing the dataset with above functions inside.

After the data are loaded and converted into numpy arrays, it is now ready for data preparation stage by converting it into HDF5 file format. The following functions perform this step:

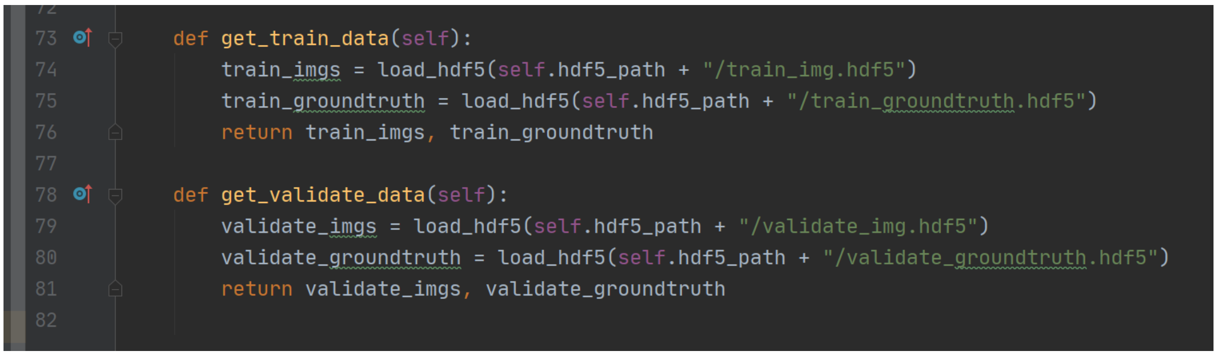


```python
    def prepare_dataset(self):
        train_imgs, groundtruth = self.access_dataset(self.train_img_path, self.train_groundtruth_path)
        write_hdf5(train_imgs, self.hdf5_path + "/train_img.hdf5")
        write_hdf5(groundtruth, self.hdf5_path + "/train_groundtruth.hdf5")
        print("[INFO] Saving Training Data...")

        validate_imgs, groundtruth = self.access_dataset(self.validate_img_path, self.validate_groundtruth_path)
        write_hdf5(validate_imgs, self.hdf5_path + "/validate_img.hdf5")
        write_hdf5(groundtruth, self.hdf5_path + "/validate_groundtruth.hdf5")
        print("[INFO] Saving Validation Data...")

        print("[INFO] Data Saved")
```

Figure 25. Code snippet 4 - showing the functions for saving numpy arrays of train, validate set into HDF5 files.

The train and validate set will be saved into the HDF5 folders, where each set are separated into 2 HDF5 files consisting of images and ground-truths, resulting in 4 HDF5 files containing metadata of train images/ground-truths and validate images/ground-truths respectively.

DataLoader class is also responsible for getting the transformed HDF5 data files. The two functions are responsible for loading the train data and validate data in HDF5, and respectively, return the images and its corresponding ground-truths. The code snippet below shows the getter functions in the DataLoader class:

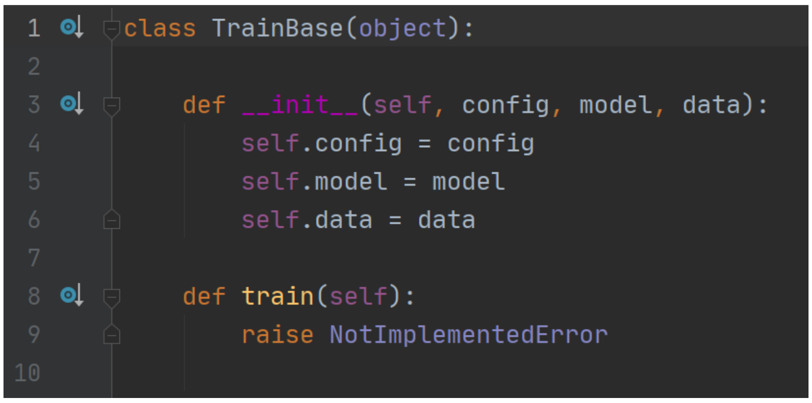


```python
    def get_train_data(self):
        train_imgs = load_hdf5(self.hdf5_path + "/train_img.hdf5")
        train_groundtruth = load_hdf5(self.hdf5_path + "/train_groundtruth.hdf5")
        return train_imgs, train_groundtruth

    def get_validate_data(self):
        validate_imgs = load_hdf5(self.hdf5_path + "/validate_img.hdf5")
        validate_groundtruth = load_hdf5(self.hdf5_path + "/validate_groundtruth.hdf5")
        return validate_imgs, validate_groundtruth
```

Figure 26. Code snippet 5 - Train and validate data getter in DataLoader class.

5.2.3 Train class

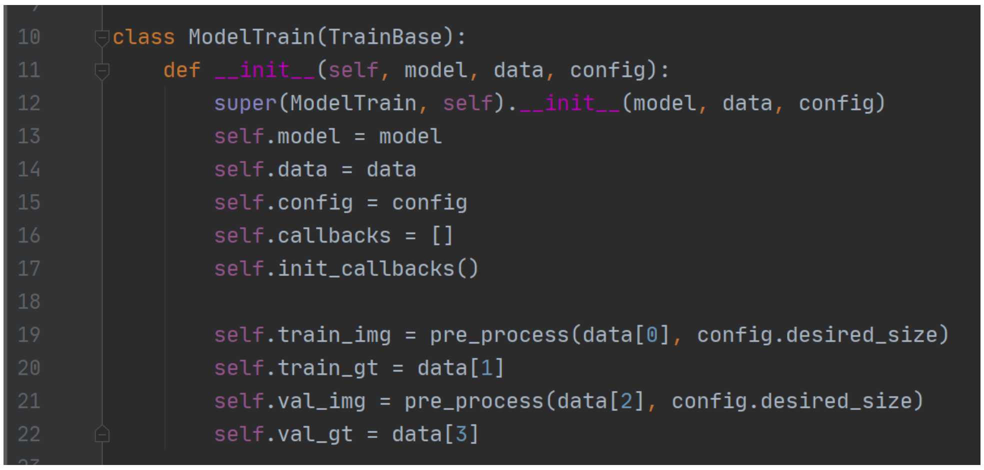
The TrainBase parent class is the base class for training the model. This class takes in json configuration file, the model, and all the necessary data for training. The following code snippet shows the implementation of TrainBase parent class.



```python
class TrainBase(object):

    def __init__(self, config, model, data):
        self.config = config
        self.model = model
        self.data = data

    def train(self):
        raise NotImplementedError

```

Figure 27. Code snippet 6 – TrainBase class

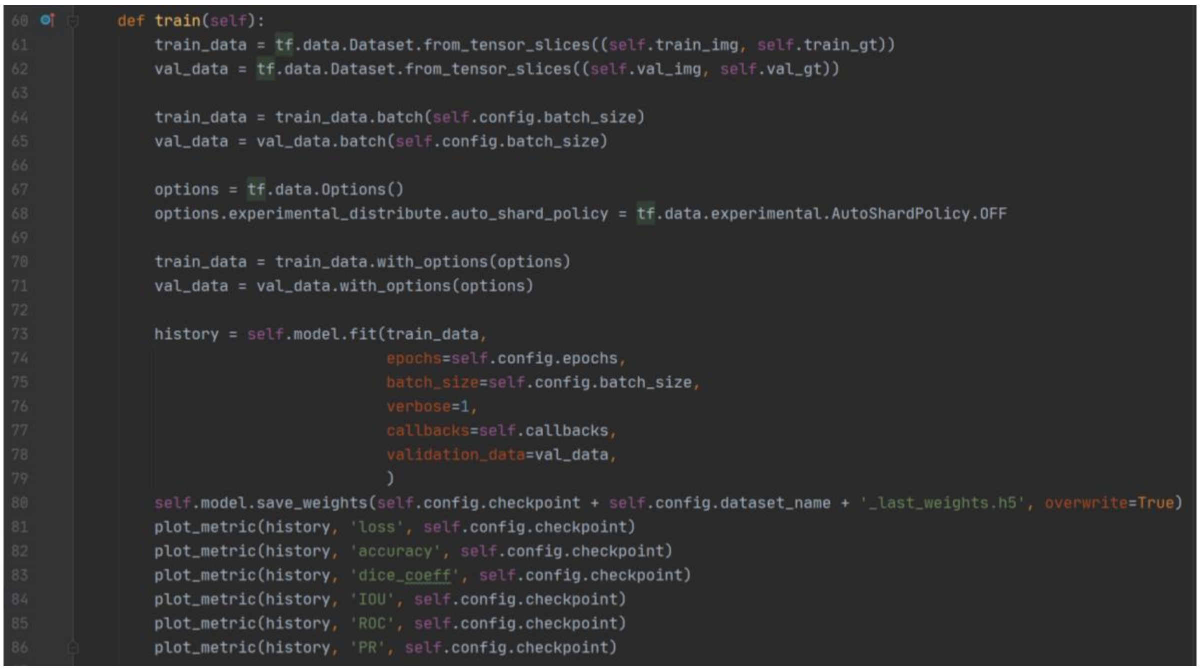
This parent class is inherited by the ModelTrain class whom is responsible for training of the model. As usual, the ModelTrain class takes in the same parameters as its parent class. In addition to this, it also initialize callbacks list in order to keep the list of callbacks that will be used in the training. The following code snippet shows the initialized variables when the class is called:



```python
class ModelTrain(TrainBase):
    def __init__(self, model, data, config):
        super(ModelTrain, self).__init__(model, data, config)
        self.model = model
        self.data = data
        self.config = config
        self.callbacks = []
        self.init_callbacks()

        self.train_img = pre_process(data[0], config.desired_size)
        self.train_gt = data[1]
        self.val_img = pre_process(data[2], config.desired_size)
        self.val_gt = data[3]
```

Figure 28. Code snippet 7 - ModelTrain class inherited from TrainBase class.

Moreover, as this class is responsible for training of the model, the data will also be passed through at the initialization of the class. This data is a list consisting of train images, train ground-truths, validate images, and validate ground-truths, respectively. The images in the numpy array are automatically pre-processed and passed over to the train function down below:



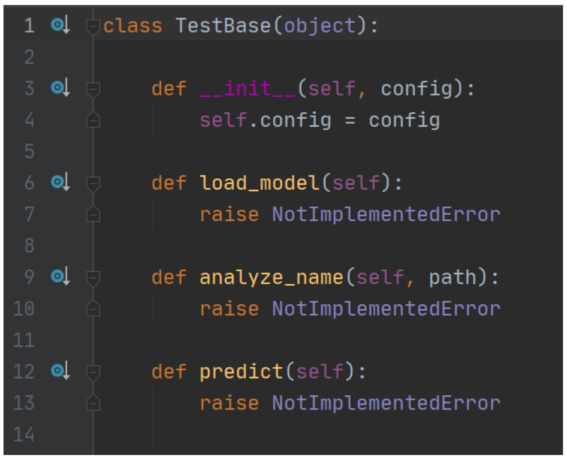
```
    def train(self):
        train_data = tf.data.Dataset.from_tensor_slices((self.train_img, self.train_gt))
        val_data = tf.data.Dataset.from_tensor_slices((self.val_img, self.val_gt))

        train_data = train_data.batch(self.config.batch_size)
        val_data = val_data.batch(self.config.batch_size)

        options = tf.data.Options()
        options.experimental_distribute.auto_shard_policy = tf.data.experimental.AutoShardPolicy.OFF

        train_data = train_data.with_options(options)
        val_data = val_data.with_options(options)

        history = self.model.fit(train_data,
                                 epochs=self.config.epochs,
                                 batch_size=self.config.batch_size,
                                 verbose=1,
                                 callbacks=self.callbacks,
                                 validation_data=val_data,
                                 )
        self.model.save_weights(self.config.checkpoint + self.config.dataset_name + '_last_weights.h5', overwrite=True)
        plot_metric(history, 'loss', self.config.checkpoint)
        plot_metric(history, 'accuracy', self.config.checkpoint)
        plot_metric(history, 'dice_coeff', self.config.checkpoint)
        plot_metric(history, 'IOU', self.config.checkpoint)
        plot_metric(history, 'ROC', self.config.checkpoint)
        plot_metric(history, 'PR', self.config.checkpoint)
```

Figure 29. Code snippet 8 - Train function for fitting the model.

In Tensorflow, tensors can be easily understood as multidimensional array just like ndarrays and is a generalization of vectors and matrices. In this project, the conversion of numpy arrays to tensors is preferred. The train and validate numpy arrays concatenated together into two dataset through the 'from_tensor_slices' function which takes in the input and its label. The data flowed into the model as batches in which it is specified in the batch function of the Dataset. After that the data are fed into the model and the training begins. While training, the best weight will be saved based on validate loss. At the end, the latest weights will be saved as well. The metrics for evaluating and for producing the graphs are plotted and saved into the checkpoint folder in the dataset folder. This will allow us to evaluate the model and discuss the results in Chapter 8 Evaluation.

5.2.4 Test class

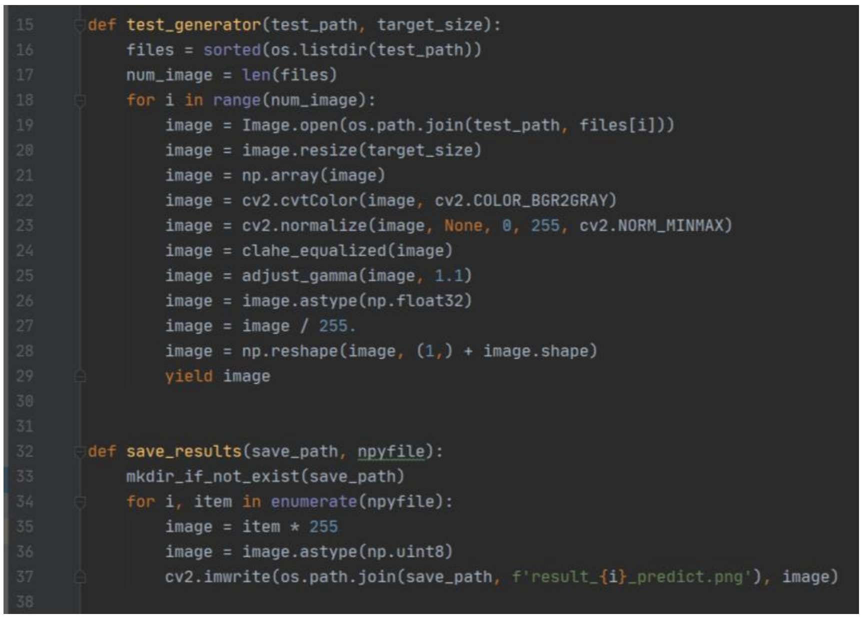
The TestBase parent class is the base class for training the model. The class takes in configuration from the config.json file. The following code snippet shows the TestBase class:



```python
class TestBase(object):

    def __init__(self, config):
        self.config = config

    def load_model(self):
        raise NotImplementedError

    def analyze_name(self, path):
        raise NotImplementedError

    def predict(self):
        raise NotImplementedError

```

Figure 30. Code snippet 9 - TrainBase class.

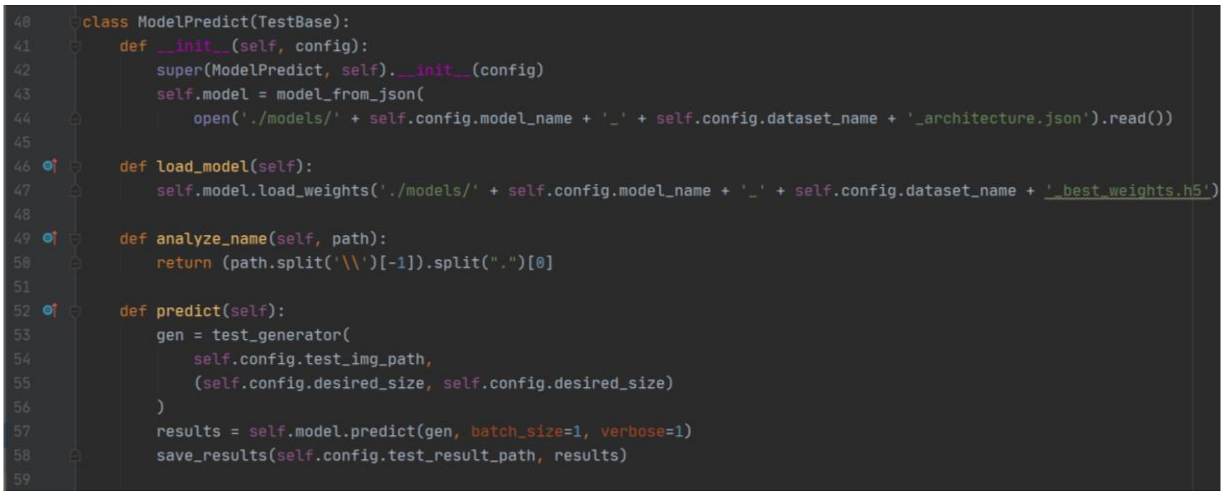
The test class is responsible for testing the model on the test data. Thus, the function for loading the model and predicting the segmentation map is needed. These are all inherited by the test class in which the testing is done. The test data will be read in using the test_generator function which yield the test image for prediction. The following code snippet shows the generator and the saving function:



```python
def test_generator(test_path, target_size):
    files = sorted(os.listdir(test_path))
    num_image = len(files)
    for i in range(num_image):
        image = Image.open(os.path.join(test_path, files[i]))
        image = image.resize(target_size)
        image = np.array(image)
        image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        image = cv2.normalize(image, None, 0, 255, cv2.NORM_MINMAX)
        image = clahe_equalized(image)
        image = adjust_gamma(image, 1.1)
        image = image.astype(np.float32)
        image = image / 255.
        image = np.reshape(image, (1,) + image.shape)
        yield image


def save_results(save_path, npyfile):
    mkdir_if_not_exist(save_path)
    for i, item in enumerate(npyfile):
        image = item * 255
        image = image.astype(np.uint8)
        cv2.imwrite(os.path.join(save_path, f'result_{i}_predict.png'), image)

```

Figure 31. Code snippet 10 - Test generator and save functions in test.py

The test class resides in the test.py where the implementation of the loading the model and doing the prediction is done. After loading the model, the best trained model weights are loaded into the model. Finally, the test image generator is called inside the predict function, flowing one image at a time, and the result is saved into the result folder. The following code snippet shows the implementation of the test class:

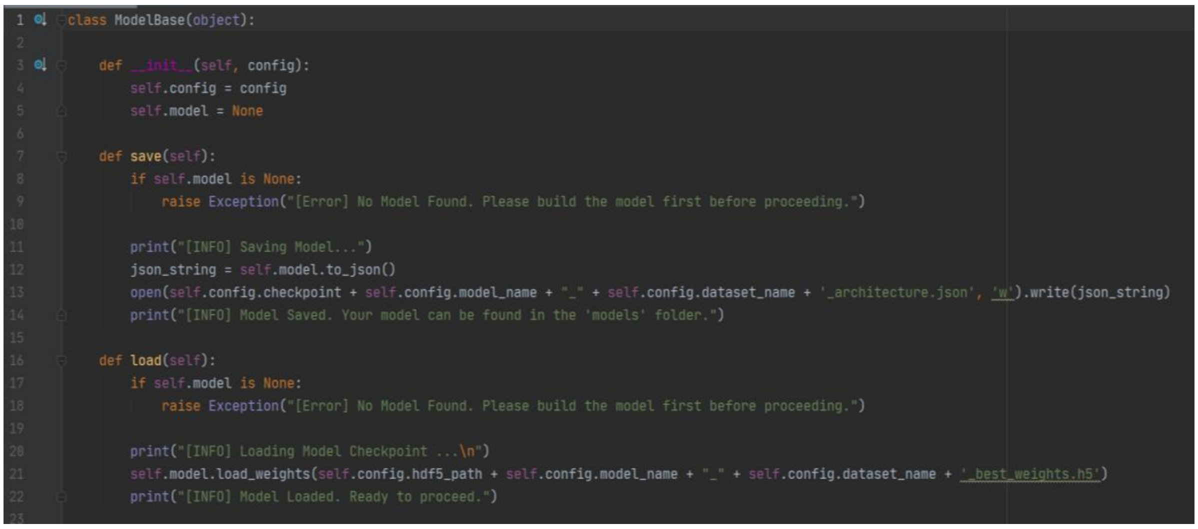


```python
class ModelPredict(TestBase):
    def __init__(self, config):
        super(ModelPredict, self).__init__(config)
        self.model = model_from_json(
            open('./models/' + self.config.model_name + '_' + self.config.dataset_name + '_architecture.json').read())

    def load_model(self):
        self.model.load_weights('./models/' + self.config.model_name + '_' + self.config.dataset_name + '_best_weights.h5')

    def analyze_name(self, path):
        return (path.split('\\')[-1]).split(".")[0]

    def predict(self):
        gen = test_generator(
            self.config.test_img_path,
            (self.config.desired_size, self.config.desired_size)
        )
        results = self.model.predict(gen, batch_size=1, verbose=1)
        save_results(self.config.test_result_path, results)
```

Figure 32. Code snippet 11 - Test class

5.3 CNN Model – U-Net

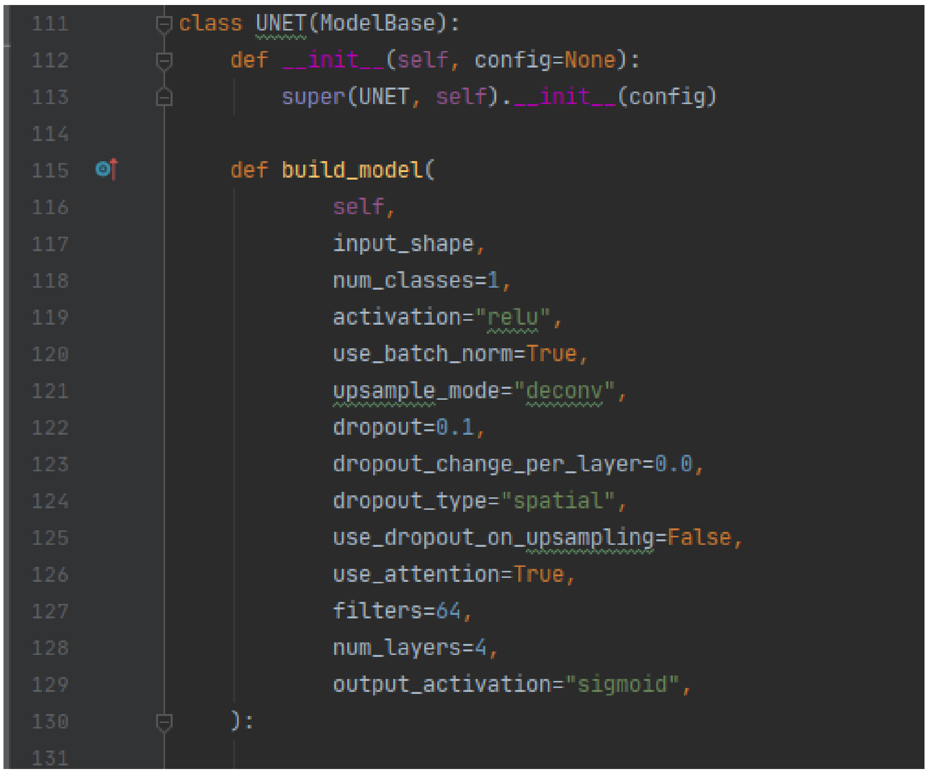
The U-Net follows the design as described in 4.4 Model Design. The U-Net class is a child class of the parent class called ModelBase.



```python
class ModelBase(object):

    def __init__(self, config):
        self.config = config
        self.model = None

    def save(self):
        if self.model is None:
            raise Exception("[Error] No Model Found. Please build the model first before proceeding.")

        print("[INFO] Saving Model...")
        json_string = self.model.to_json()
        open(self.config.checkpoint + self.config.model_name + "_" + self.config.dataset_name + '_architecture.json', 'w').write(json_string)
        print("[INFO] Model Saved. Your model can be found in the 'models' folder.")

    def load(self):
        if self.model is None:
            raise Exception("[Error] No Model Found. Please build the model first before proceeding.")

        print("[INFO] Loading Model Checkpoint ...\n")
        self.model.load_weights(self.config.hdf5_path + self.config.model_name + "_" + self.config.dataset_name + '_best_weights.h5')
        print("[INFO] Model Loaded. Ready to proceed.")
```

Figure 33. Code snippet 12 - ModelBase class.

Since this is the model class, it is important for any implementation concerning the machine learning models. Thus, the functions – save and load – were implemented into the parent model class. In the UNET class, the build_model function is inherited for later implementation of the CNN model. The parameter of the build_model function has all the changeable parameter for modifying the model's layers. The following code snippet shows the parameters of the build_model class that was inherited into UNET class.

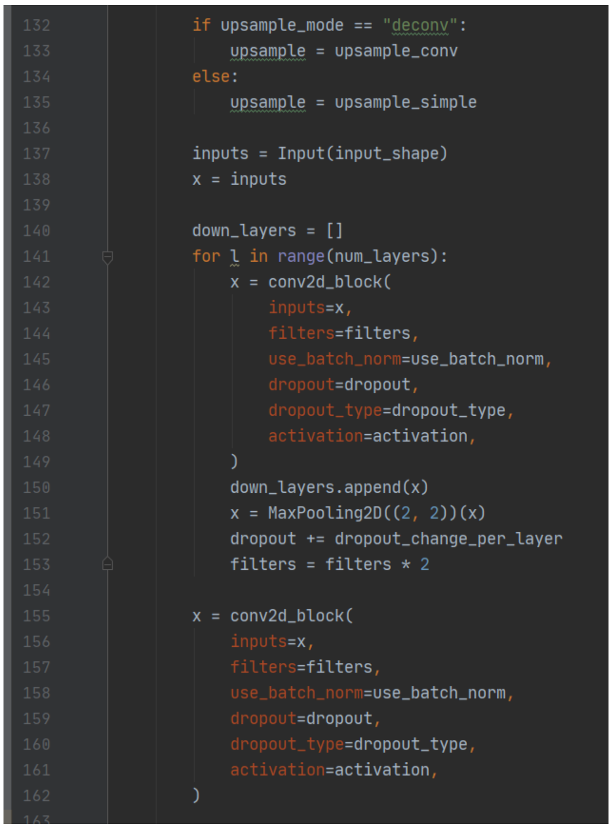


```python
class UNET(ModelBase):
    def __init__(self, config=None):
        super(UNET, self).__init__(config)

    def build_model(
            self,
            input_shape,
            num_classes=1,
            activation="relu",
            use_batch_norm=True,
            upsample_mode="deconv",
            dropout=0.1,
            dropout_change_per_layer=0.0,
            dropout_type="spatial",
            use_dropout_on_upsampling=False,
            use_attention=True,
            filters=64,
            num_layers=4,
            output_activation="sigmoid",
    ):

```

Figure 34. Code snippet 13 – UNET class with parameters of build_model function shown

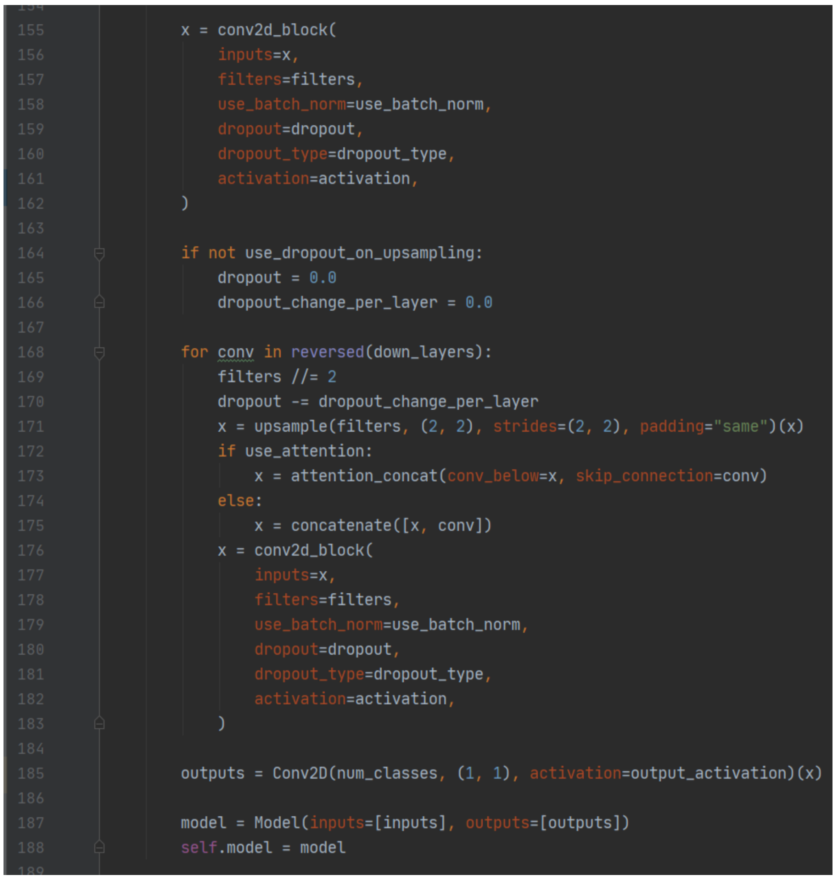
As because we want to be able to modify the U-Net without interfering with the actual implementation of fully-connected convolutional layers, the layers and the functions used inside the layers were implemented as functions outside the UNET class such as attention_gate, because they are specific to each model architecture, thus the UNET class is only responsible in putting everything together and build the model.

The for-loop is used for building the architecture of the UNET because, as mentioned in 4.4 Model Design, the U-Net architecture consists of contracting path and expanding path, in which their only difference is the downsampling for contraction and upsampling for expanding. Thus, it is easier to use for-loop for building both side. The following code snippets show how the implementation of building the model was done:



```python
        if upsample_mode == "deconv":
            upsample = upsample_conv
        else:
            upsample = upsample_simple

        inputs = Input(input_shape)
        x = inputs

        down_layers = []
        for l in range(num_layers):
            x = conv2d_block(
                inputs=x,
                filters=filters,
                use_batch_norm=use_batch_norm,
                dropout=dropout,
                dropout_type=dropout_type,
                activation=activation,
            )
            down_layers.append(x)
            x = MaxPooling2D((2, 2))(x)
            dropout += dropout_change_per_layer
            filters = filters * 2

        x = conv2d_block(
            inputs=x,
            filters=filters,
            use_batch_norm=use_batch_norm,
            dropout=dropout,
            dropout_type=dropout_type,
            activation=activation,
        )
```

Figure 35. Code snippet 14 - The contracting path of U-Net



```python
        x = conv2d_block(
            inputs=x,
            filters=filters,
            use_batch_norm=use_batch_norm,
            dropout=dropout,
            dropout_type=dropout_type,
            activation=activation,
        )

        if not use_dropout_on_upsampling:
            dropout = 0.0
            dropout_change_per_layer = 0.0

        for conv in reversed(down_layers):
            filters //= 2
            dropout -= dropout_change_per_layer
            x = upsample(filters, (2, 2), strides=(2, 2), padding="same")(x)
            if use_attention:
                x = attention_concat(conv_below=x, skip_connection=conv)
            else:
                x = concatenate([x, conv])
            x = conv2d_block(
                inputs=x,
                filters=filters,
                use_batch_norm=use_batch_norm,
                dropout=dropout,
                dropout_type=dropout_type,
                activation=activation,
            )

        outputs = Conv2D(num_classes, (1, 1), activation=output_activation)(x)

        model = Model(inputs=[inputs], outputs=[outputs])
        self.model = model
```

Figure 36. Code snippet 15 - The expanding path of U-Net

After the U-Net model is built with the specified parameter, the next step is to compile the model to prepare it for training. The parameters of the compile_model function allows the U-Net optimizer, learning rate, loss function, and metrics for modification.

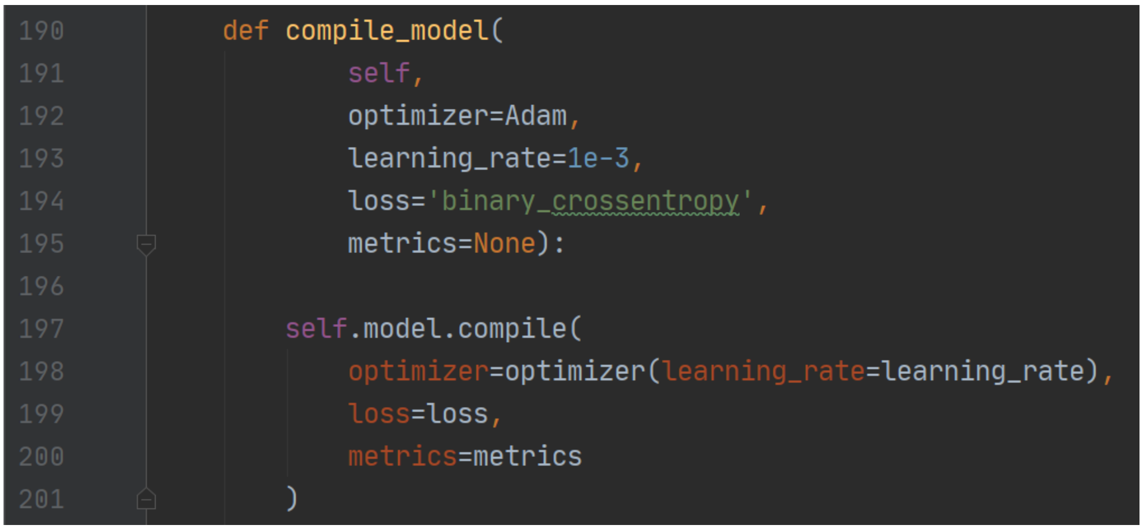

```
    def compile_model(
            self,
            optimizer=Adam,
            learning_rate=1e-3,
            loss='binary_crossentropy',
            metrics=None):

        self.model.compile(
            optimizer=optimizer(learning_rate=learning_rate),
            loss=loss,
            metrics=metrics
        )
```

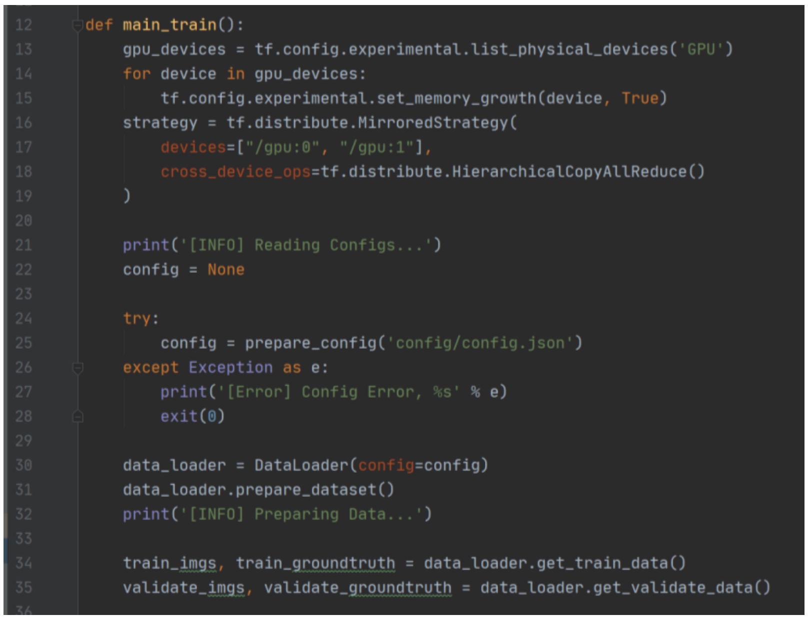
Figure 37. Code snippet 16 - compile_model function inside UNET class

5.4 Model Training and Testing

Now that all the necessary pipeline and model implementation are done, it is time to put them to use. There are two main scripts in this project which are main_train.py and main_test.py. As the name suggested, in main_train.py, this script runs the training of the model, and in main_test.py, this performs the prediction of the segmentation map.

5.4.1 Model Training – main_train.py

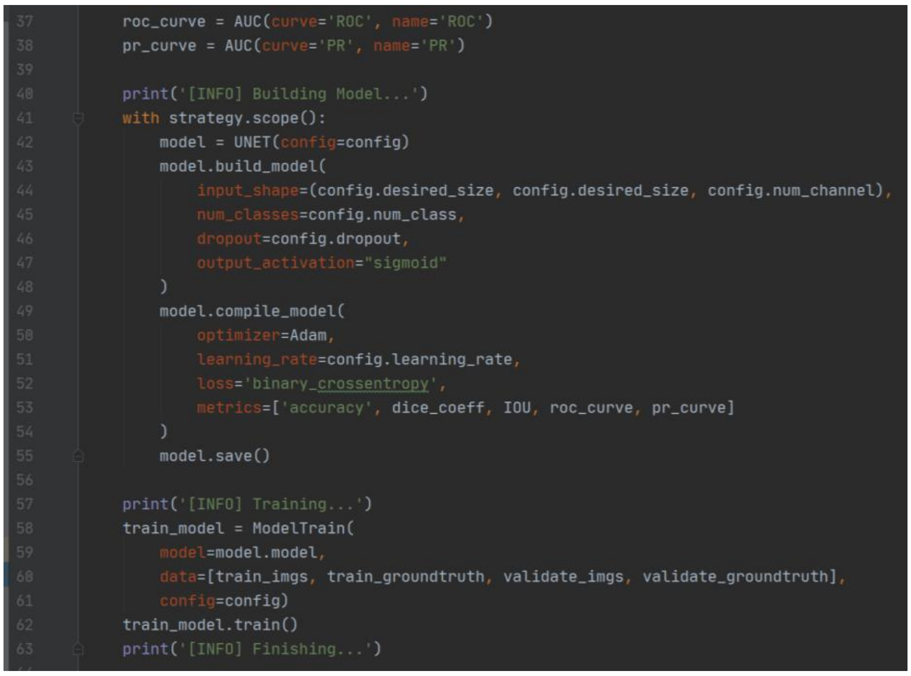
In this main file, the training of the model is performed here. This script only contains one function called main_train. A configuration is also needed and that is the parameters set inside the json config file in which it specifies the dataset name, model name, batch size, learning rate etc. All of the parameters set are stored inside a variable called config, in which it will be passed into other classes called in this script. After loading the config, it is time to run the data pipeline. The variable data_loader is the class object of DataLoader class which calls the function prepare_dataset in order to convert the data and save it in HDF5 format. After that we get the data and store it as variables. This step is shown in the code snippet below:



```python
def main_train():
    gpu_devices = tf.config.experimental.list_physical_devices('GPU')
    for device in gpu_devices:
        tf.config.experimental.set_memory_growth(device, True)
    strategy = tf.distribute.MirroredStrategy(
        devices=["/gpu:0", "/gpu:1"],
        cross_device_ops=tf.distribute.HierarchicalCopyAllReduce()
    )

    print('[INFO] Reading Configs...')
    config = None

    try:
        config = prepare_config('config/config.json')
    except Exception as e:
        print('[Error] Config Error, %s' % e)
        exit(0)

    data_loader = DataLoader(config=config)
    data_loader.prepare_dataset()
    print('[INFO] Preparing Data...')

    train_imgs, train_groundtruth = data_loader.get_train_data()
    validate_imgs, validate_groundtruth = data_loader.get_validate_data()
```

Figure 38. Code snippet 17 - Main Script for Training - Part 1.

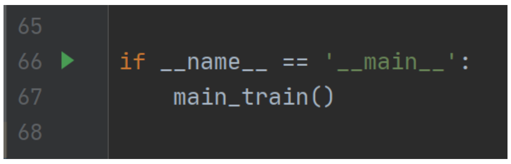
The second part of the script performs the build and compile of the model and call the train function to begin training. From the top, the metrics that will be used in the training are initialize and input it into the model when compiled. The model is initialized as an object of UNET class. The desired size is the size that the images will be resized to. Although it can be resized to any size, however, to achieve good performance of U-Net model, the desired size has to be divisible by 32, thus a good size such as 512 × 512 is good.



```python
    roc_curve = AUC(curve='ROC', name='ROC')
    pr_curve = AUC(curve='PR', name='PR')

    print('[INFO] Building Model...')
    with strategy.scope():
        model = UNET(config=config)
        model.build_model(
            input_shape=(config.desired_size, config.desired_size, config.num_channel),
            num_classes=config.num_class,
            dropout=config.dropout,
            output_activation="sigmoid"
        )
        model.compile_model(
            optimizer=Adam,
            learning_rate=config.learning_rate,
            loss='binary_crossentropy',
            metrics=['accuracy', dice_coeff, IOU, roc_curve, pr_curve]
        )
        model.save()

    print('[INFO] Training...')
    train_model = ModelTrain(
        model=model.model,
        data=[train_imgs, train_groundtruth, validate_imgs, validate_groundtruth],
        config=config)
    train_model.train()
    print('[INFO] Finishing...')
```

Figure 39. Code snippet 18 - Main Script for Training - Part 2.

After the model is built, we compile the model and save it in json form in the checkpoint folder. The train object of ModelTrain class takes in model, list of data, and the config file, and then the training start when the train object called the train function. Finally, the training can begin when this script is run in the command terminal.

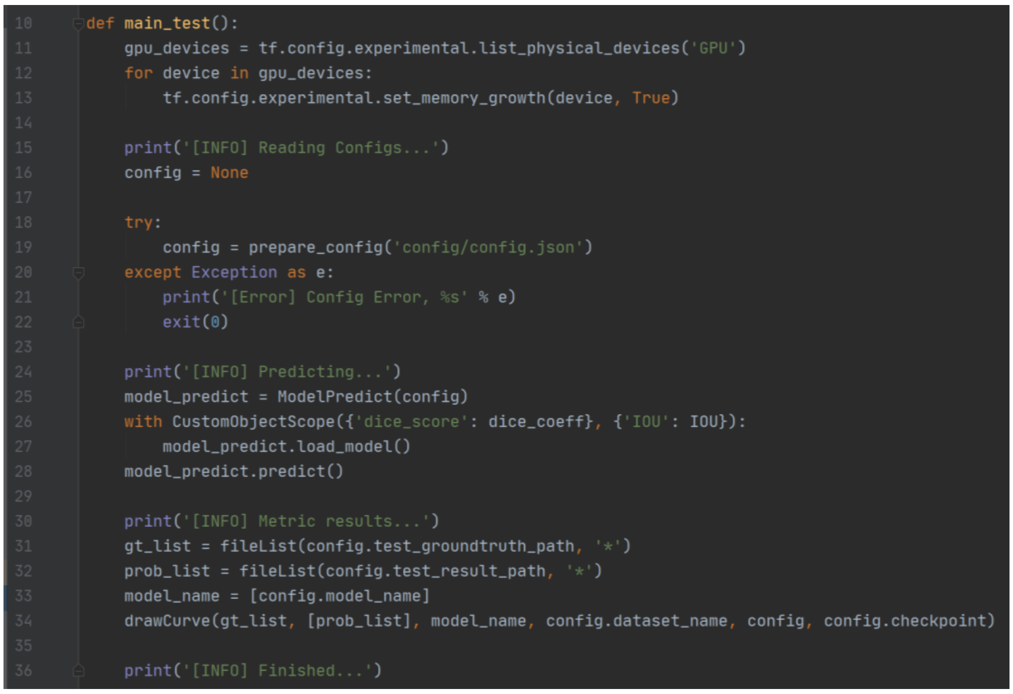


```python
if __name__ == '__main__':
    main_train()
```

Figure 40. Code snippet 19 - run training when main_train.py script is called.

5.4.2 Model Test – main_test.py

In this main file, the testing/predicting is performed here. This scripts contains only one function which is called main_test, and it is called when this script run in the terminal. The model_predict object then initialized as the object of ModelPredict class that takes in the config variable. The function load_model then loads the model in json form and initialized the weights with the best weights of the trained model. Since we trained with custom metrics, the Tensorflow requires us to load and compile the model with its special Class that takes in our custom metrics. Subsequently, we can begin the testing of our model.



```python
def main_test():
    gpu_devices = tf.config.experimental.list_physical_devices('GPU')
    for device in gpu_devices:
        tf.config.experimental.set_memory_growth(device, True)

    print('[INFO] Reading Configs...')
    config = None

    try:
        config = prepare_config('config/config.json')
    except Exception as e:
        print('[Error] Config Error, %s' % e)
        exit(0)

    print('[INFO] Predicting...')
    model_predict = ModelPredict(config)
    with CustomObjectScope({'dice_score': dice_coeff}, {'IOU': IOU}):
        model_predict.load_model()
    model_predict.predict()

    print('[INFO] Metric results...')
    gt_list = fileList(config.test_groundtruth_path, '*')
    prob_list = fileList(config.test_result_path, '*')
    model_name = [config.model_name]
    drawCurve(gt_list, [prob_list], model_name, config.dataset_name, config, config.checkpoint)

    print('[INFO] Finished...')
```

Figure 41. Code snippet 20 - main_test.py

When the testing finished, the functions gt_list and prob_list creates two lists that keeps the actual ground-truths and the predicted output, respectively. This is then feed into the drawCurve function which performs the necessary calculations to plot the PR and ROC curve and save it into the checkpoint folder for further evaluation.

After all going through all the process, the logging of the steps are saved in the checkpoint folder. This folder keeps the graphs and weights produced after the training stage as well as the testing stage.

5.5 Summary

In this chapter, we first looked at the overall project structure as well as the data pipeline necessary in performing the evaluation in the next chapter. The components of the structure were described on its purposes and the linkage between them, justifying the use of OOP approach.

All of the significant components in the solution used to achieve the aim were shown, subsequently, displayed the natural flow of progress from structuring the project to loading and processing the data to building the model, and to using the model. By joining the components together, they allow the model to perform training and prediction which ultimately, provide output segmentation of the blood vessels and optic disc.

This chapter achieves objective 5 by implementing the system as specified in Chapter 4 Design. The chapter shows how each of the core components of implementation were done and how it relates back to the aim of this study.

6 Experiments and Evaluations

6.1 Retinal Vessel Segmentation

In the process of diagnosing the retinal diseases in the fundus images, the retinal blood vessels segmentation is an important step in diagnosis. This is because the blood vessels width, diameter and such also allows the ophthalmologist and eye doctor to determine the health state of the patient. For example, if there are abnormal growth of blood vessels in the retina, the eye can be categorised into proliferative diabetic retinopathy stage.

6.1.1 Materials

The DRIVE dataset will be used in this problem. This publicly available dataset consists of total of TIFF 40 colour fundus images; including 7 abnormal pathology cases. Each image resolution is 584x565 pixels with eight bits per colour channel (3 channels). The dataset of 40 images was equally divided into 20 images for the training set and 20 images for the testing set. We chose to use the first observer segmentation as it is accepted by the community as the golden standard.

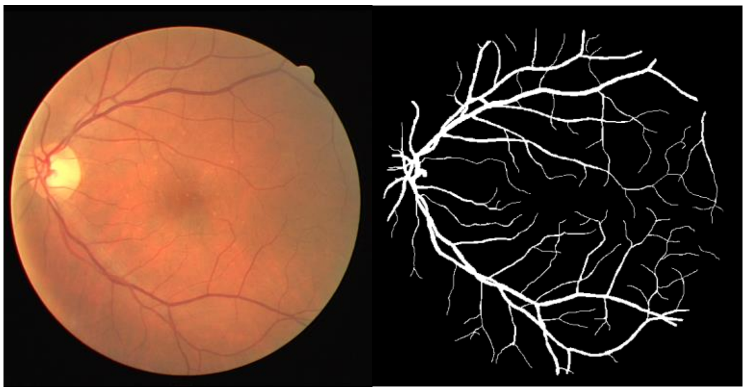
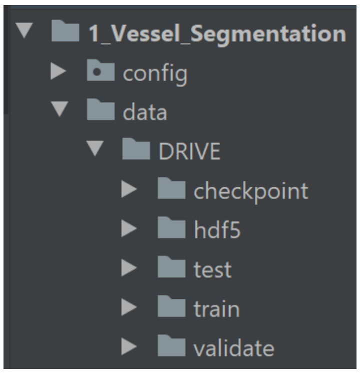


Figure 42. An example of a retinal image with the manual segmentation of blood vessels in DRIVE dataset.

Since the dataset was originally split into train and test set, the training set was further split into training and validate set in order to have validation set during the training of the model.



```
▼ 1_Vessel_Segmentation
    ▶ config
    ▼ data
        ▼ DRIVE
            ▶ checkpoint
            ▶ hdf5
            ▶ test
            ▶ train
            ▶ validate
```

Figure 43. Data folder structure with DRIVE dataset split into train, validate, and test sets

6.1.2 Data Augmentation

There is one obvious issue that this project is facing and that is the shortage of annotated blood vessels of fundus images. Considering that DRIVE has 40 images in total, and only 20 images were used for training, this can lead to insufficient number of training samples in which typically, deep learning based models will likely suffer from over-fitting problem. Thus, data augmentation is a solution to this shortage problem. In this project, two ways of increasing training samples will be done.

Image augmentation

The common solution that were applied to the training samples are: random rotation, colour jittering, random Gaussian blur, and image flipping. This made the number of training samples to increase from 20 images to 232 images, resulting in reduction of over-fitting of the model.

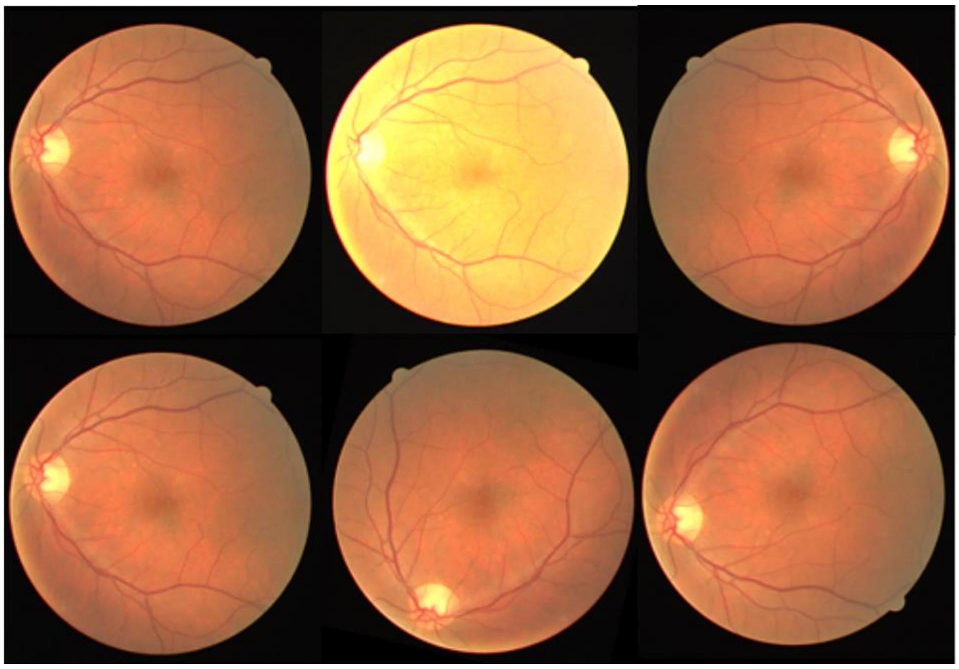


Figure 44. From top left to top right: (1) Original image, (2) Random colour applied, (3) Horizontal flip applied. From bottom left to bottom right: (4) Random Gaussian blur applied, (5) Random rotation applied, (6) Vertical flip applied.

6.1.3 Pre-process

It is worth noting that detecting the blood vessels is a regression task. It seems unnecessary to use colour information because the colours may just add extra complexity. For this reason, all of the images were converted to greyscale for use. For the purpose of this study, the images were resized to 512x512 pixels and its corresponding vessel ground-truth mask were scaled accordingly. The contrast of the resized images was enhanced by applying the contrast-limited adaptive histogram equalisation technique (CLAHE) so as to reduce uneven illumination in the images as shown in Figure 3.1. The pixel values of the enhanced images were converted to 32-bit floating point and scaled between [0,1] floating points. Code snippet below shows pre-processing utilities.

Figure below shows the pre-processed image in each step of pre-processing.

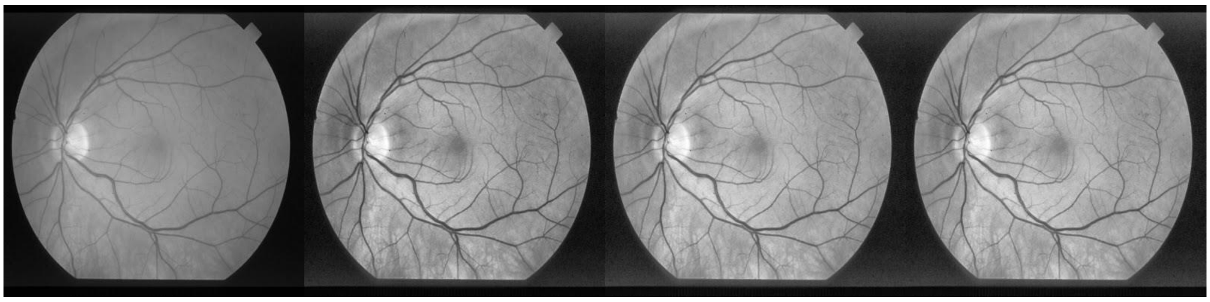


Figure 45. Pre-processed retinal image in each pre-processing stage. From left to right: (1) the grey-scaled image of the retinal image, (2) CLAHE applied to the image, (3) gamma adjustment of the image, (4) the image is normalized into [0,1] range.

6.1.4 Training and Testing

One of the technique to approach this is patch-based model which cut the whole image into patches and feed it into the neural network. However, since U-Net can perform segmentation in less than a second on an 512 × 512 image with recent GPU, thus this led the model used in this experiment to be an image-based model in which it takes in an image as a whole image.

Despite that, obtaining best results when training a network relies on the correct selection of hyperparamers. As there are many possible hyperparameters to select and not all the possible combinations can be tried, after some experiments ones with the best outcomes were chosen. The following table shows the hyperparamets used in building the model in this experiment:

Table 2. Hyperparamets in build model

Batch Norm	Dropout	Dropout Type	Attention Gates
True	0.2	Spatial	True

Batch size is set to 4 and the learning rate to 0.001 (1e-3). Different optimizers have been tested to see which one obtains fit better to our problems. However, best results were all obtained with Adam optimizer algorithms. The reason for choosing Adam optimizer as the optimization algorithm was because it is a very useful algorithms that can handle sparse gradients on noisy problems due to its being the combination of the best properties of the AdaGrad and RMSProp algorithms. Thus, it was chosen to be experimented.

The following table shows the hyperparameters used in training the model:

Table 3. Hyperparameters in model training

Epochs	Batch Size	Optimizer	Learning Rate	Loss Function
40	4	Adam	0.001	Binary Cross Entropy

We trained the model for 40 epochs using batch size of 4, using binary cross-entropy as loss function and Adam as our optimizer with learning rate of 0.001 (1e-3). By using the hyperparameters above to train the model, we are able to obtain graphs based on the metrics we used. In Figure 45 shows metrics on train and validate set which are: Loss, Accuracy, Dice Score, and IoU. The loss function we used in this experiment is binary cross-entropy. The following graph is the loss value of train and validate plotted against the number of epochs:

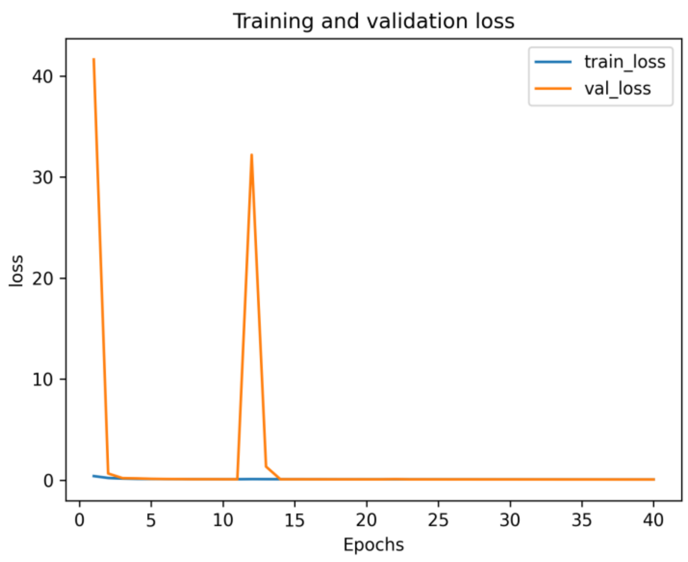


Figure 46. Loss VS Number of Epochs

We can see that the accuracy of both training set and validating set is almost the same, in which train accuracy is at 97% and validation accuracy is at 97.36%. This indicates that the model is neither over-fitting nor under-fitting, in which this shows the robustness of the model that can generalize well when faced with unseen dataset.

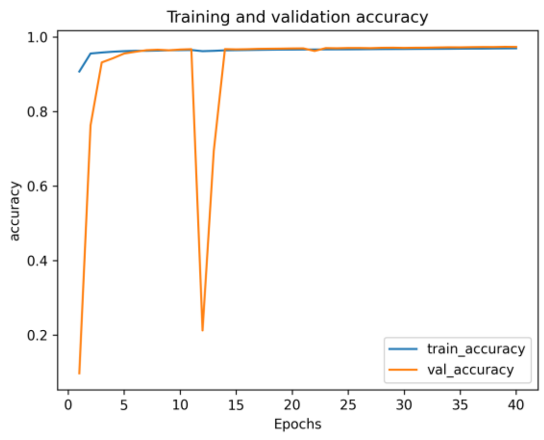


Figure 47. Accuracy VS Number of Epochs

As mentioned in 3.3.1 Metrics, AUC was used in as a metric for evaluating the discrimination ability of our model. The following figure of AUC graph shows the value of AUC over time:

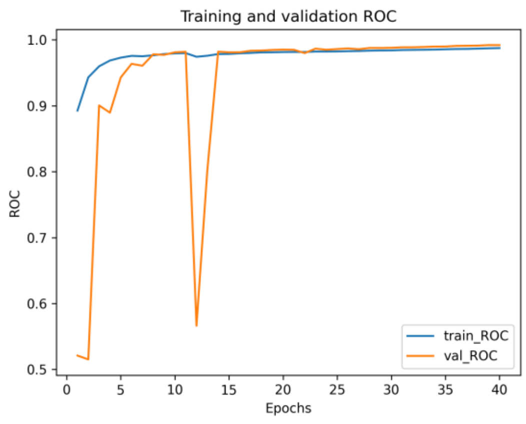


Figure 48. AUC VS Number of Epochs

Another metric that was also used in the Dice coefficient metric is used as described in Chapter 3 Methodology. The following figure of Dice coefficient graphs shows the value of Dice coefficient trained over time:

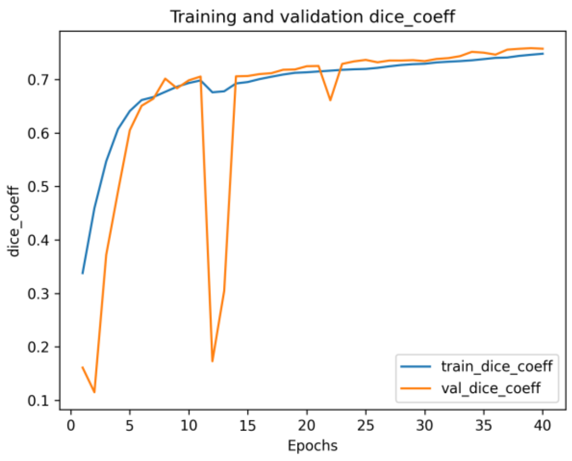


Figure 49. Dice score VS Number of Epochs

As we can see in the figure Dice coefficient vs No. of epochs above, the Dice coefficients increases as the number of epochs increases, this shows a good sign of the model that the predicted output 'overlaps' correctly with the ground-truth.

The following table shows the validation set results from the final epoch:

Table 4. Final result of training on DRIVE dataset.

Model	Loss	Accuracy	AUC	Dice coefficient
U-Net without DA	0.0855	0.9674	0.9793	0.7066
U-Net with DA	0.0631	0.9687	0.9836	0.7576

U-Net = Final modified U-Net model, DA = Data Augmentation.

Finally, we tested the model on the test data which comprises of 20 images that we held out at the start of the experiment. The following table shows the results in each metrics evaluated on the test set:

Table 5. Results on DRIVE dataset.

Model	Accuracy	AUC	Dice coefficient
U-Net without DA	0.9653	0.9764	0.6994
U-Net with DA	0.9705	0.9831	0.7398

U-Net = Final modified U-Net model, DA = Data Augmentation.

And these are the final results of segmentation using the trained modified U-Net with data augmentation:

Figure 50. Results on DRIVE dataset.

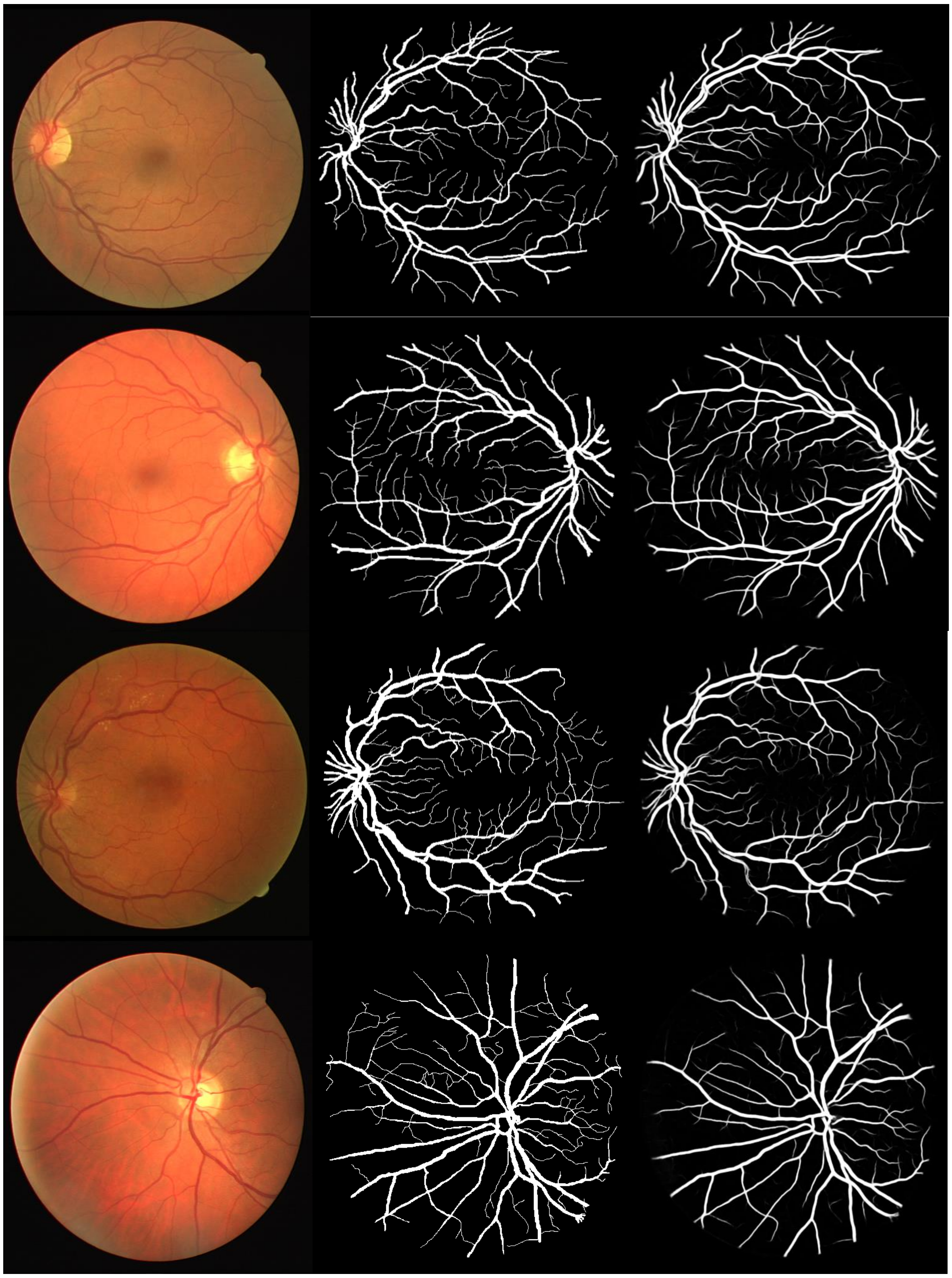


Figure 51. Segmentation results from DRIVE dataset. Left column: Original Image. Middle column: Ground-truth. Right column: Segmented output

6.2 Optic Disc Segmentation

In the process of diagnosing the retinal diseases in the fundus images, the retinal optic disc segmentation is part of the diagnosis. This is because optic disc is associated with a retinal disease which is Glaucoma. Thus, the segmentation will also help the ophthalmologist and eye doctor in diagnosing the condition of the optic disc in the retina faster.

6.2.1 Materials

The DRIONS-DB dataset will be used in this experiment. This publicly available dataset consists of total of JPEG 110 colour fundus images. The images were acquired with a colour analogical fundus camera. Each image resolution is 600 × 400 pixels with eight bits per colour channel (3 channels). The first set of annotations was chosen as this is accepted as the golden standard for performance evaluation. An original image and its corresponding ground-truth can be seen below:

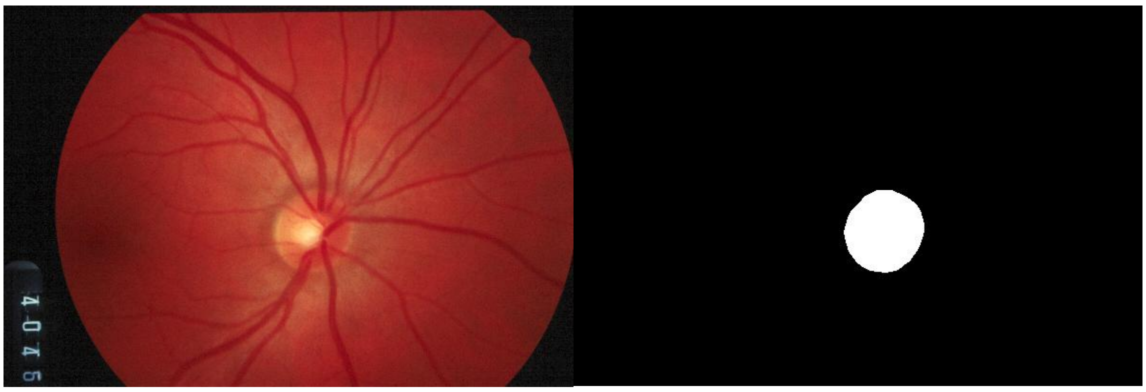


Figure 52. An example of a retinal image with the manual segmentation of optic disc in DRIONS-DB dataset.

6.2.2 Data Augmentation

The data augmentation performed on DRIVE dataset was also performed on DRIONS-DB dataset. Augmentations include random rotation, colour jittering, random Gaussian blur, and image flipping.

6.2.3 Pre-process

For the same reasons as the retinal vessels segmentation, there is no need for colour as it adds the unnecessary complexity, thus all of the images were converted to greyscale for use. All the pre-processing is the same as in retinal vessels images.

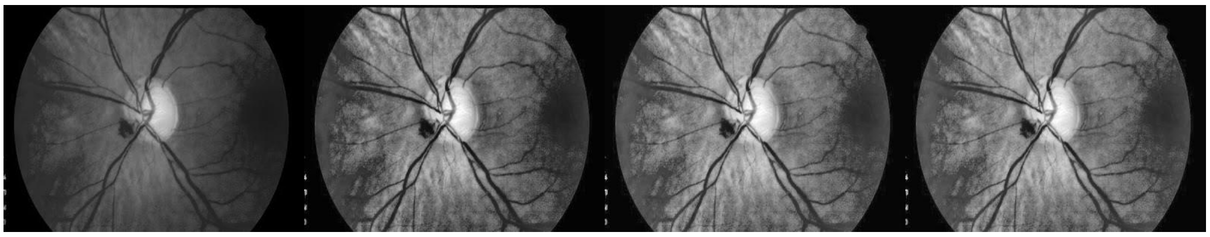


Figure 53. Pre-processed retinal image in each pre-processing stage. From left to right: (1) the grey-scaled image of the retinal image, (2) CLAHE applied to the image, (3) gamma adjustment of the image, (4) the image is normalized into [0,1] range.

6.2.4 Training and Testing

The following table shows the hyperparamets used in building the model in this experiment:

Table 6. Hyperparamets in model building.

Batch Norm	Dropout	Dropout Type	Attention Gates
True	0.2	Spatial	True

Table 7. Hyperparameters in model training

Epochs	Batch Size	Optimizer	Learning Rate	Loss Function
40	4	Adam	0.001	Binary Cross Entropy

We trained the model for 40 epochs using batch size of 4, using binary cross-entropy as loss function and Adam as our optimizer with learning rate of 0.001 (1e-3). The metrics used in this experiment is also the same, thus the graphs produced are the same metrics.

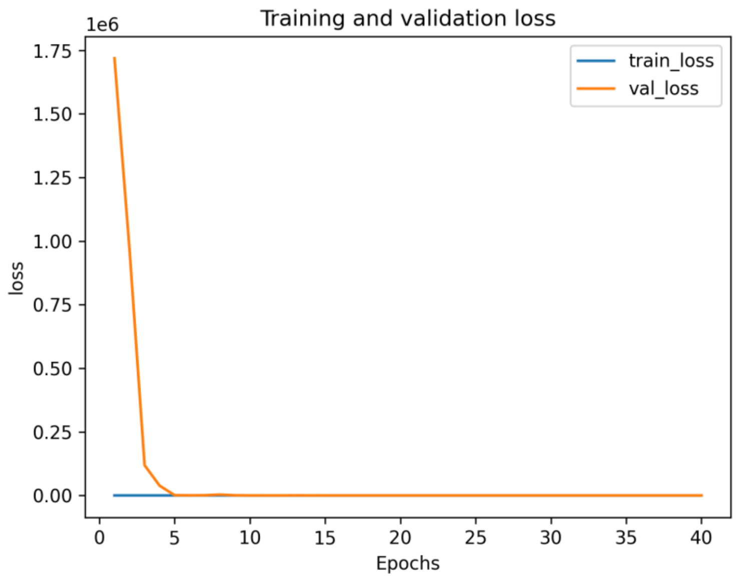


Figure 54. Loss VS Number of Epochs

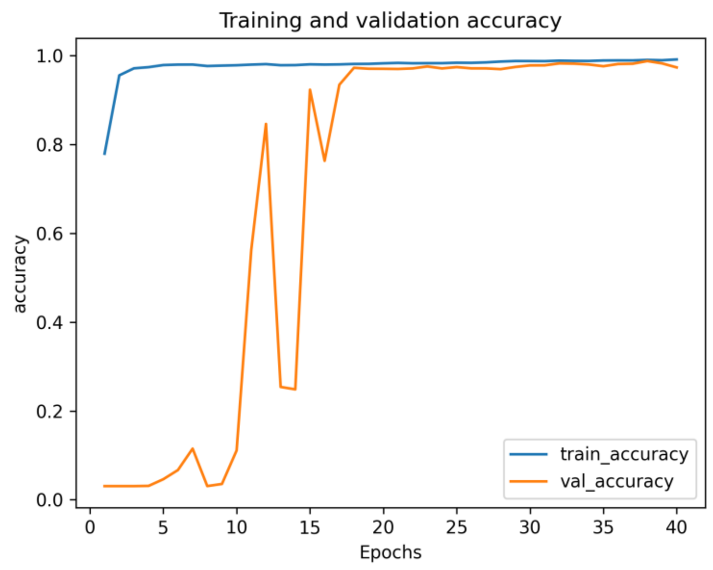


Figure 55. Accuracy VS Number of Epochs

In Figure 57, the validate accuracy fluctuated by a lot but it still increases in which it stops fluctuating and flowed together with training accuracy.

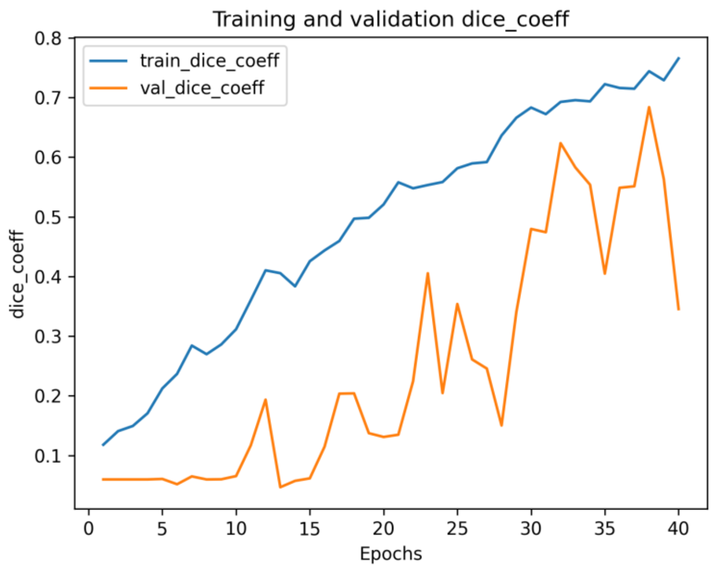


Figure 56. Dice coefficient VS Number of Epochs

As we can see in the graph above, the Dice coefficient of training set climbed peacefully, meanwhile, although the Dice coefficient of validate set does increase, however, it fluctuates up and down as it goes through the epochs. This may be an indicator of our model over-fitting.

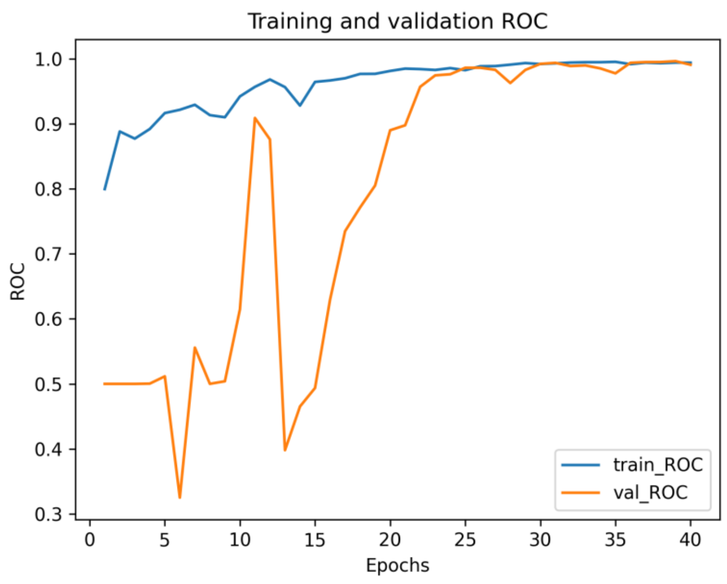


Figure 57. AUC VS Number of Epochs

Table 8. Final result of training on DRIONS-DB dataset.

Model	Loss	Accuracy	AUC	Dice coefficient
U-Net with DA	0.0649	0.9733	0.9907	0.3454

U-Net = Final modified U-Net model, DA = Data Augmentation.

Finally, we tested the model on the test data which comprises of 19 images that we held out at the start of the experiment. The following table shows the results in each metrics evaluated on the test set:

Table 9. Results on DRIONS-DB dataset.

Model	Accuracy	AUC	Dice coefficient
U-Net with DA	0.9546	0.9732	0.418

U-Net = Final modified U-Net model, DA = Data Augmentation.

With consideration of both experiments tested, the accuracy of both retinal vessel and optic disc segmentation were all above 95%. This means that our modified U-Net can generalize well on more than one dataset that have no common characteristics. In which this also means that our model is robust enough to achieve a satisfying result on an unseen dataset. Furthermore, the AUC values were all above 95% as well, which means that our U-Net model has the ability to discriminate the pixels correctly. On the other hand, the Dice coefficient is surprisingly low. One explanation could be that this metric is not suited for our experiments. Another explanation would be the number of epochs are too low which causes the model to not be able to learn properly. From the results we obtained, the modified U-Net are capable and reliable that can produce a good quality segmentation map with an average time of less than a second, however, due to its inaccuracy, it would not be a viable model to be used in clinical applications.

6.3 Summary

In this chapter, we have experimented, conducted, and reported the evaluation results of the model. The results from training our model for 40 epochs with batch size of 4 in both experiments were obtained and discussed throughout the chapter. Comparisons of the ground-truth and the predicted output of our model were shown as well in order to evaluate the model qualitatively. Ultimately, our model developed in this work would not be practical because its accuracy is not high enough when it comes to real medical world applications. This chapter achieves objective 7 and 8 by conducting test and produced graphical representation of the results in which the results were further evaluated.

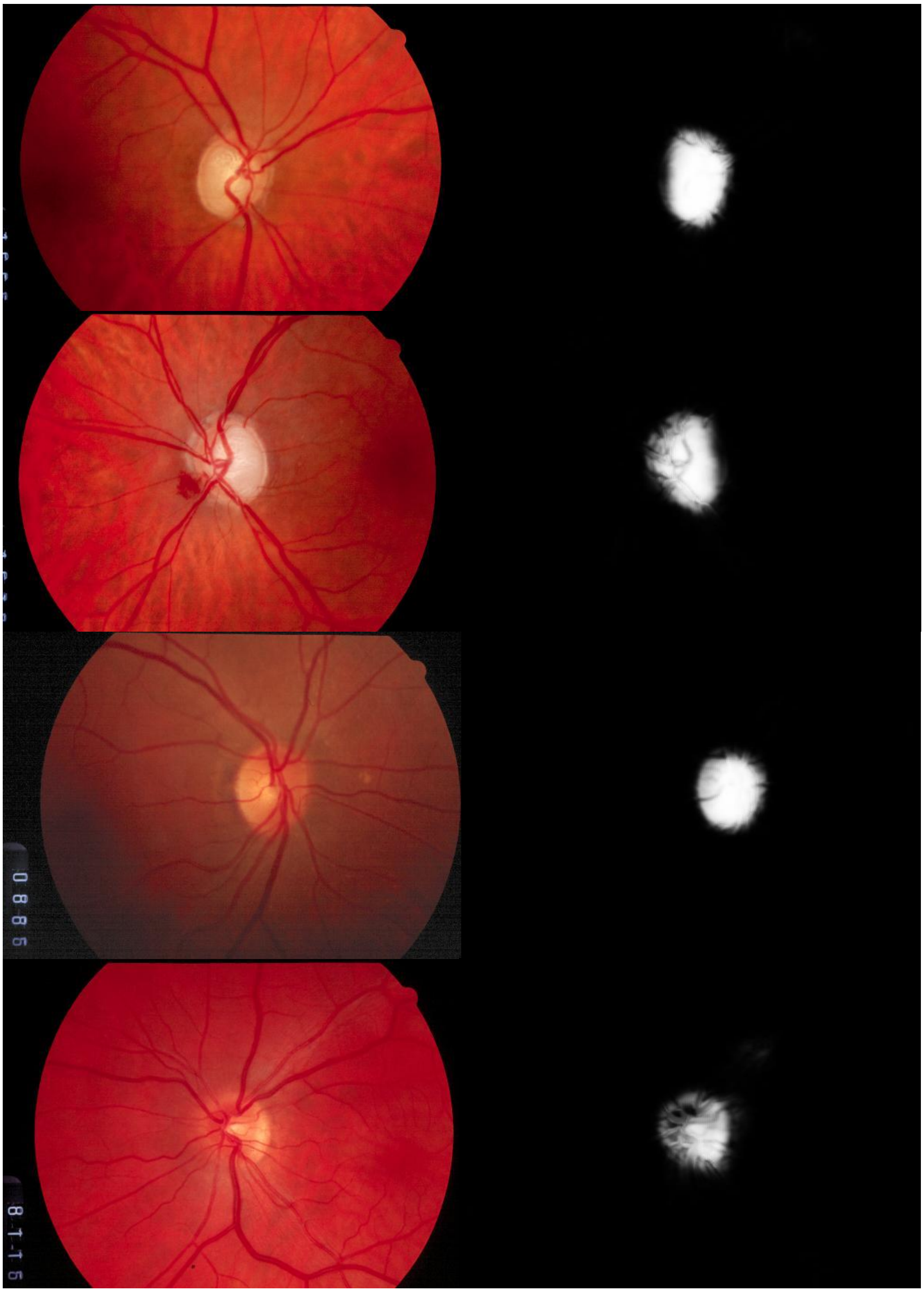


Figure 58. Results from segmenting optic disc on DRIONS-DB dataset.

7 Conclusions

7.1 Project Overview

To summarise, the goal of this project is to address the issue of time cost in screening patients' eye and manual segmentation as well as the workload on the healthcare workers by developing a CNN Model in order to automate the segmentation process of retinal images. We found that a U-Net model would be most suitable which can be adapted and improved upon. This is then led us to developing a system solution which is based on reproducibility, scalability, and generalizability. The final solution was a system with natural flow in data pipeline and utilized the modified U-Net to learn on new dataset as well as predicting on unseen data. The performance of the model was then measured and discussed in the evaluation process.

In my opinion, the structure of the system implemented was good and produced a maintainable functionally working system. However, I think that the research methodology could have been more detailed and more in-depth since it is the chapter where we presents our ideas and how we are going to approach for the solution. Possibly, different machine learning models could be experimented and make comparisons on how effective it can solve this problem. The evaluation process in this project could then be used to determine which one is better suited.

7.2 Meeting the Aims and Objectives

As explained in 1 Introduction, this dissertation focused on the addressing the issue of time cost and workload on performing diagnosis of the retinal conditions. This dissertation aimed to determine if using deep learning algorithms – CNNs - in learning the data can produce a satisfying segmentation map on the similar data will be able to reduce the workload on healthcare workers and helps them in performing the task faster.

The aim of the project was met through the development of the model and software solution that utilized CNN model to train on vessels and optic disc segmentation and segment them out in new images. Upon evaluation of the aim against the model, it was found that the solution developed would not be a practical solution for healthcare workers to use to address the issue of time cost and workload.

Table 10. Objectives table.

No.	Objective	How it was met?

1	Explore existing retinal image segmentation techniques and implementations by undertaking relevant research, and also through the analysis of applicable background literature.	Through the literature review, it was found that there are many techniques in the field of image processing such as low-level image processing and high-level image processing in which these algorithms can segment the retinal image too. Also, it was found out that there are many deep learning algorithms that were applied in various fields of medical image processing and segmentation. An overview of existing image segmentation was also summarized and taken into consideration during the experiments.
2	Based on the research carried out through objective 1, identify which (if any) methods could be adopted and further developed to provide an effective, reliable solution for the problem of retinal image segmentation.	It was found out that U-Net has recently become extensively used in the field of image segmentation in which many newly adapted or enhanced version of U-Net were proposed, achieving at state-of-the-art results. Thus, U-Net was chosen to be adapted into the project in which modifications and improvements were done upon it.
3	Before moving on to design and development of automated retinal image segmentation, the acquisition of labelled datasets consisting of marked structures of blood vessels and optic disc.	The datasets were identified through the public website and established it as the data that we will benchmark on. The labelled data were acquired through the data source on public site. Two datasets were confirmed and be concentrated on in the later stages.
4	After acquiring the necessary datasets and identifying a feasible solution through objective 2, design and develop CNN model based on the models and techniques adapted from the feasible solution.	The research methodology determines the mechanisms presented in new model which were chosen to be integrated into the design. This objective was met through bringing practices discovered from experience and research into the project. Certain promising mechanisms such as Attention gates, Spatial dropout were recognized and fed into the design of the U-Net CNN model.

5	After planning and designing the system architecture and CNN model in objective 4, develop and implement the data pipeline and ultimately implement the automated image processing and segmentation.	This objective was met through the implementation of the system. The implementation were created based on the framework and designs produced in objective 2 and 4. The implementation details have been explained fully with a natural flow from one sub-section to the other.
6	Train the model through supervised learning, leveraging the datasets provided corresponding to each stage of segmentation tasks.	This objective was met in conducting the experiments by using the datasets discovered in objective 3. In the experiments, the model were trained with different hyper-parameters, ultimately settling for the best one.
7	Conduct testing of the model on the test data. Important statistics and graphs such as accuracy or Dice score, will be produced to determine and validate the performance of the model in the model evaluation.	This objective was met in the same section as the objective 6. Both tabular and graphical representations of the results were produced and presented with description on each one of them.
8	After thoroughly testing the model, the result produced in object 7 will be evaluated to determine the performance of the CNN model.	This last objective was met after all the objectives were gone through from 1 to 7, in which after testing the model, the comparisons were made and the evaluation process was done as described in chapter 3.

7.3 Future Work

The following table provides the directions for future work for the software solution developed.

Table 11. Future work table.

Possible Future work	Description

Further development of the modified U-Net	As mentioned in 6.1.4 Training/Testing, the hyper-parameter settings such as the regularization strength, the learning rate, the optimizer, and other hyper-parameters were set manually for training after some experimenting were done. Therefore, the model could be further developed by using an automated hyper-parameter search method such as GridSearch.
Further development of the system	As demonstrated in 5.2 System Architecture, the system architecture has only one layer in which the whole system stays on. The system could be further developed with Graphical User Interface (GUI) and implemented in a way that the user can upload their own retinal image and the system automatically segment it.
Further Evaluation and Validation of the solution	As mentioned in 3.2.2 Data Augmentation, the size of the training samples are very small in which these data points can be validate and evaluate further. For example, k-fold cross validation which is very suitable for small datasets like what we used in the experimentation. Furthermore, the evaluation could go further by performing a clinical evaluation of the developed approaches which could allow us to assess its feasibility in clinical ophthalmology.

Bibliography

[1] Jampol, L., Shankle, J., Schroeder, R., Tornambe, P., Spaide, R. and Hee, M., 2006. Diagnostic and Therapeutic Challenges. *Retina*, [online] 26(9), pp.1072-1076. Available at: < https://journals.lww.com/retinajournal/Citation/2006/11000/Diagnostic_and_Therapeutic_Challenges.15.aspx> [Accessed 3 March 2021].

[2] Tufail, A., Rudisill, C., Egan, C., Kapetanakis, V., Salas-Vega, S., Owen, C., Lee, A., Louw, V., Anderson, J., Liew, G., Bolter, L., Srinivas, S., Nittala, M., Sadda, S., Taylor, P. and Rudnicka, A., 2017. Automated Diabetic Retinopathy Image Assessment Software. *Ophthalmology*, [online] 124(3), pp.343-351. Available at: <https://www.aaojournal.org/article/S0161-6420(16)32018-8/fulltext> [Accessed 3 March 2021].

[3] Niemeijer, M., Abràmoff, M. and van Ginneken, B., 2009. Fast detection of the optic disc and fovea in color fundus photographs. *Medical Image Analysis*, 13(6), pp.859-870.

[4] Acharya, U., Dua, S., Du, X., Sree S, V. and Chua, C., 2011. Automated Diagnosis of Glaucoma Using Texture and Higher Order Spectra Features. *IEEE Transactions on Information Technology in Biomedicine*, [online] 15(3), pp.449-455. Available at: <https://ieeexplore.ieee.org/document/5720314> [Accessed 3 March 2021].

[5] Acharya, U., Ng, E., Eugene, L., Noronha, K., Min, L., Nayak, K. and Bhandary, S., 2015. Decision support system for the glaucoma using Gabor transformation. *Biomedical Signal Processing and Control*.

[6] Giancardo, L., Meriaudeau, F., Karnowski, T., Li, Y., Garg, S., Tobin, K. and Chaum, E., 2012. Exudate-based diabetic macular edema detection in fundus images using publicly available datasets. *Medical Image Analysis*, [online] 16(1), pp.216-226. Available at: <https://pubmed.ncbi.nlm.nih.gov/21865074/> [Accessed 3 March 2021].

[7] Giancardo, L., Meriaudeau, F., Karnowski, T., Li, Y., Garg, S., Tobin, K. and Chaum, E., 2012. Exudate-based diabetic macular edema detection in fundus images using publicly available datasets. *Medical Image Analysis*, [online] 16(1), pp.216-226. Available at: <https://pubmed.ncbi.nlm.nih.gov/21865074/> [Accessed 3 March 2021].

[8] Aquino, A., 2014. Establishing the macular grading grid by means of fovea centre detection using anatomical-based and visual-based features. *Computers in Biology and Medicine*, [online] 55, pp.61-73. Available at: <https://pubmed.ncbi.nlm.nih.gov/25450220/> [Accessed 3 March 2021].

[9] Soto-Pedre, E., Navea, A., Millan, S., Hernaez-Ortega, M., Morales, J., Desco, M. and Pérez, P., 2014. Evaluation of automated image analysis software for the detection of diabetic retinopathy to reduce the ophthalmologists' workload. *Acta Ophthalmologica*, [online] 93(1), pp.e52-e56. Available at: <https://pubmed.ncbi.nlm.nih.gov/24975456/> [Accessed 3 March 2021].

[10] Veiga, D., Pereira, C., Ferreira, M., Gonçalves, L. and Monteiro, J., 2014. Quality evaluation of digital fundus images through combined measures. *Journal of Medical Imaging*.

[11] Suzuki, K., 2017. Overview of deep learning in medical imaging. *Radiological Physics and Technology*, [online] 10(3), pp.257-273. Available at: <https://pubmed.ncbi.nlm.nih.gov/28689314/> [Accessed 3 March 2021].

[12] Kels, B., Grzybowski, A. and Grant-Kels, J., 2015. Human ocular anatomy. *Clinics in Dermatology*, [online] 33(2), pp.140-146. Available at: <https://pubmed.ncbi.nlm.nih.gov/25704934/> [Accessed 5 March 2021].

[13] Remington, L., 2012. *Clinical Anatomy and Physiology of the Visual System*. pp.61-92.

[14] Oyster, C.W., 1999. *The human eye: structure and function*. Sinauer Associates.

[15] Diabetes. 2019. Diabetic retinopathy usually only affects people who have had diabetes (diagnosed or undiagnosed) for a significant number of years. *Diabetes.co.uk* [online] Available at: <https://www.diabetes.co.uk/diabetes-complications/diabetic-retinopathy.html> [Accessed 6 March 2021].

[16] Tarr, J., Kaul, K., Chopra, M., Kohner, E. and Chibber, R., 2013. Pathophysiology of Diabetic Retinopathy. *ISRN Ophthalmology*, [online] 2013, pp.1-13. Available at: <https://pubmed.ncbi.nlm.nih.gov/24563789/> [Accessed 7 March 2021].

[17] nhs.uk. n.d. *Glaucoma*. [online] Available at: <https://www.nhs.uk/conditions/glaucoma/> [Accessed 7 March 2021].

[18] Saine, P.J., and Tyler, M.E., 2002. *Ophthalmic photography: retinal photography, angiography, and electronic imaging*, volume 132. Butterworth-Heinemann Boston

[19] Caponetti, L. and Castellano, G., 2016. Low-Level Image Processing. *SpringerBriefs in Electrical and Computer Engineering*, pp.15-37.

[20] Pizer, S., Amburn, E., Austin, J., Cromartie, R., Geselowitz, A., Greer, T., ter Haar Romeny, B., Zimmerman, J. and Zuiderveld, K., 1987. Adaptive histogram equalization and its variations. *Computer Vision, Graphics, and Image Processing*, [online] 39(3), pp.355-368. Available at: <https://www.sciencedirect.com/science/article/abs/pii/S0734189X8780186X> [Accessed 8 March 2021].

[21] Zuiderveld, K., 1994. Contrast Limited Adaptive Histogram Equalization. *Graphics Gems*, pp.474-485.

[22] N, S. and S, V., 2016. Image Segmentation By Using Thresholding Techniques For Medical Images. Computer Science & Engineering: *An International Journal*

[23] Padmapriya, B., Kesavamurthi, T. and Ferose, H., 2012. Edge Based Image Segmentation Technique for Detection and Estimation of the Bladder Wall Thickness. *Procedia Engineering*, [online] 30, pp.828-835.

[24] Ansari, M. and Anand, R., 2007. Region based segmentation and image analysis with application to medical imaging. *IET-UK International Conference on Information and Communication Technology in Electrical Sciences (ICTES 2007)*, [online] Available at: <https://ieeexplore.ieee.org/document/4735891/> [Accessed 10 March 2021].

[25] Freedman, D., Radke, R., Tao Zhang, Yongwon Jeong, Lovelock, D. and Chen, G., 2005. Model-based segmentation of medical imagery by matching distributions. *IEEE Transactions on*

Medical Imaging, [online] 24(3), pp.281-292. Available at: <https://ieeexplore.ieee.org/document/1397816> [Accessed 10 March 2021].

[26] Lim, Y. and Lee, S., 1990. On the color image segmentation algorithm based on the thresholding and the fuzzy c-means techniques. *Pattern Recognition*, [online] 23(9), pp.935-952. Available at: <https://www.sciencedirect.com/science/article/abs/pii/003132039090103R> [Accessed 10 March 2021].

[27] Hinton, G., Osindero, S. and Teh, Y., 2006. A Fast Learning Algorithm for Deep Belief Nets. *Neural Computation*, [online] 18(7), pp.1527-1554.

[28] Salakhutdinov, R., and Hinton, G., 2009. Deep boltzmann machines. *In Artificial Intelligence and Statistics*, pages 448–455.

[29] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y., 2014. Generative adversarial nets. *In Advances in Neural Information Processing Systems*, pages 2672–2680.

[30] Sak, H., Senior, A., and Beaufays, F., 2014. Long short-term memory recurrent neural network architectures for large scale acoustic modeling. *In Fifteenth Annual Conference of the International Speech Communication Association*.

[31] Krizhevsky, A., Sutskever, I., and Hinton, G.E., 2012. Imagenet classification with deep convolutional neural networks. In Advances in Neural Information Processing Systems, pages 1097–1105.

[32] Abdel-Hamid, O., Mohamed, A., Jiang, H., Deng, L., Penn, G. and Yu, D., 2014. Convolutional Neural Networks for Speech Recognition. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, [online] 22(10), pp.1533-1545. Available at: <https://ieeexplore.ieee.org/document/6857341> [Accessed 13 March 2021].

[33] Ronneberger, O., Fischer, P. and Brox, T., 2015. U-Net: Convolutional Networks for Biomedical Image Segmentation. *Lecture Notes in Computer Science*, [online] pp.234-241. Available at: <https://arxiv.org/abs/1505.04597> [Accessed 13 March 2021].

[34] Wang, B., Qiu, S. and He, H., 2019. Dual Encoding U-Net for Retinal Vessel Segmentation. *Lecture Notes in Computer Science*, [online] pp.84-92. Available at: <https://www.researchgate.net/publication/336393112_Dual_Encoding_U-Net_for_Retinal_Vessel_Segmentation> [Accessed 13 March 2021].

[35] Guo, C., Szemenyei, M., Yi, Y., Wang, W., Chen, B., Fan, C., 2020. SA-UNet : Spatial Attention U-Net for Retinal Vessel Segmentation. [online] Available at: < https://arxiv.org/abs/2004.03696> [Accessed 13 March 2021].

[36] Zhang, S., Fu, H., Yan, Y., Zhang, Y., Wu, Q., Yang, M., and Tan, M., 2019. Attention Guided Network for Retinal Image Segmentation. Medical Image Computing and Computer Assisted Intervention – MICCAI 2019. *Lecture Notes in Computer Science*, volume 11764. Springer, Cham.

[37] Oktay et al., 2018. Attention U-Net: Learning Where to Look for the Pancreas. [online] Available at: < https://arxiv.org/abs/1804.03999> [Accessed 14 March 2021].
[38] Tompson, J., Goroshin, R., Jain, A., LeCun, Y. and Bregler, C., 2015. Efficient object localization using Convolutional Networks. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, [online] Available at: <https://ieeexplore.ieee.org/document/7298664> [Accessed 14 March 2021].
[39] Narkhede, S., 2018. *Understanding AUC - ROC Curve*. [online] Medium. Available at: <https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5> [Accessed 14 March 2021].
[40] Tiu, E., 2019. *Metrics to Evaluate your Semantic Segmentation Model*. [online] Medium. Available at: <https://towardsdatascience.com/metrics-to-evaluate-your-semantic-segmentation-model-6bcb99639aa2> [Accessed 14 March 2021].
[41] Sugimura, P., Hartl, F. 2018. *Building a Reproducible Machine Learning Pipeline*. [online] Available at: https://arxiv.org/abs/1810.04570 [Accessed 16 March 2021].
[42] 2020. *First-class Citizen*. Available at: https://en.wikipedia.org/wiki/First-class-citizen [Accessed 16 March 2021].
[43] *Hierarchical Data Format*. [online] Available at: < https://en.wikipedia.org/wiki/Hierarchical_Data_Format> [Accessed 17 March 2021].
[44] *Object-oriented programming*. [online] Available at: < https://en.wikipedia.org/wiki/Object-oriented_programming> [Accessed 17 March 2021].

Appendix A Personal Reflection

A.1 Reflection on Project

This project brought together many different aspects of the computer science field and made use of many different topics both learnt during university study and internship experience. The project made use of the fundamental programming concepts taught to build the software solution. This knowledge was further reinforced with experience from the time of my summer internship, bringing good practices not just from a maintainability and efficiency standpoint but also from a structural perspective. The practices such as the object-oriented data pipeline structure, made the solution significantly more structured and easier to extend.

In my opinion, I believe the overall project was a success, as the predefined aim and objectives were clearly met. Furthermore, the modified model which I designed and trained, demonstrated an excellent performance in segmenting both the retinal vessels and the optic disc.

However, upon reflection of the project, I have identified several minor adjustments that I would implement if undertaking the project again. For example, the hyper-parameters of the model were not fully optimized due to the changes were made randomly and settle for the best one. However, if I were to implement a grid search algorithms, I would not have to randomly select the initial hyper-parameters and slowly change it throughout the project.

A.2 Personal Reflection

Throughout the course of this project I have had many opportunity to challenge myself and develop a range of transferable skills. Furthermore, through hard-work and determination I have also been able to learn several new skills. For example, the correct use of Tensorflow. It is hard to familiarize with something new that looks complicated, however, I like experimentation and so I experimented with it a lot in which causes me to become familiarized automatically. Also by reflecting on my work-style throughout the project, I have also identified some minor issues that I would want myself to change if undertaking the project again. For instance, I like to 'jump' around the project which means that sometimes I stop working on a part and went to work on the other, leaving the first part behind until later. This is definitely not good and I should change this habit of mine.

Appendix B Appendices

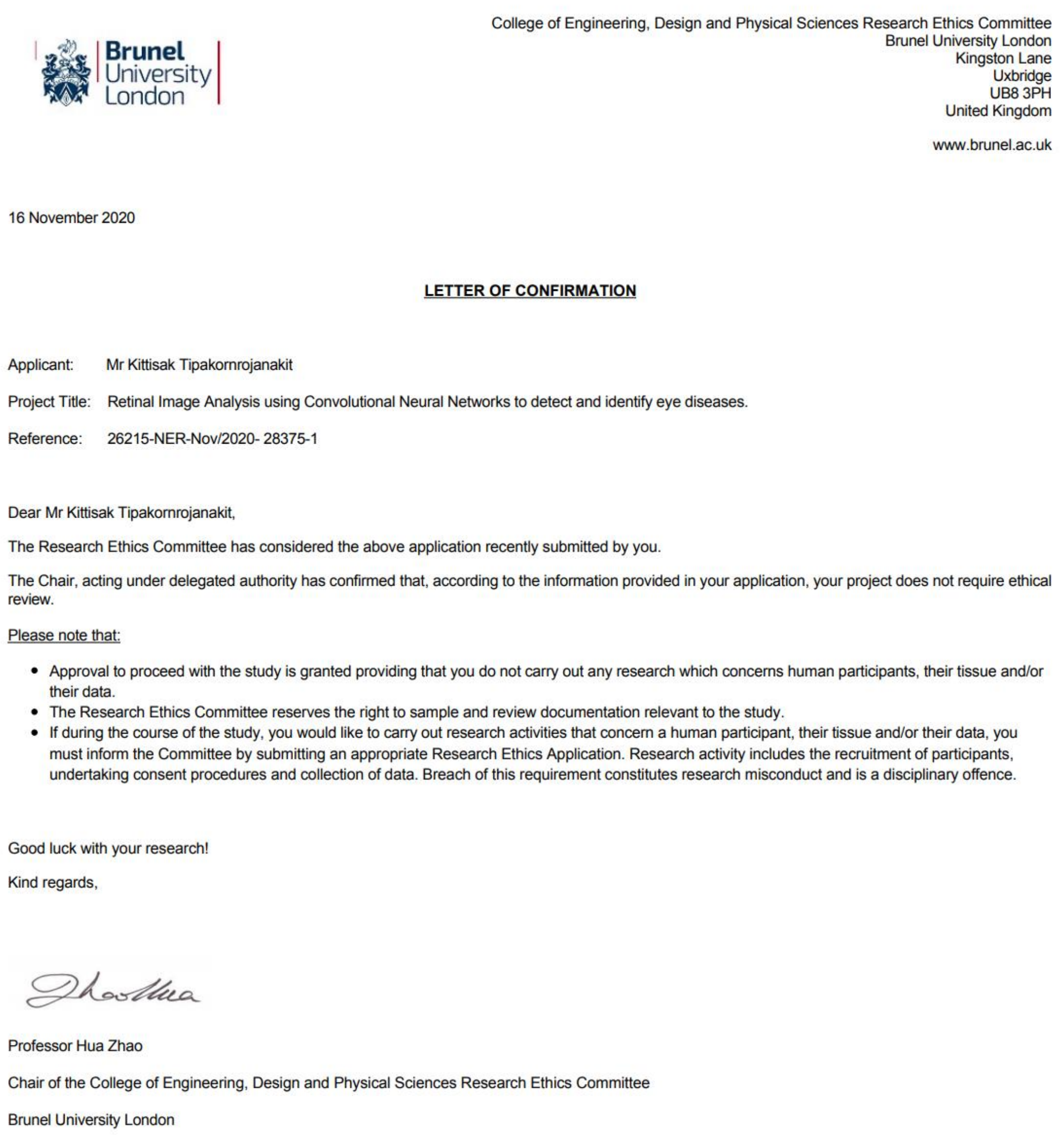
B.1 Source Codes

This appendix contains two zip files which are:

1. Vessel Segmentation
2. Optic Disc Segmentation

However weights of the model are unable to upload due to its very large size which takes a lot of time to upload.

B.2 Ethical Consideration



College of Engineering, Design and Physical Sciences Research Ethics Committee
Brunel University London
Kingston Lane
Uxbridge
UB8 3PH
United Kingdom

www.brunel.ac.uk

16 November 2020

LETTER OF CONFIRMATION

Applicant: Mr Kittisak Tipakornrojanakit

Project Title: Retinal Image Analysis using Convolutional Neural Networks to detect and identify eye diseases.

Reference: 26215-NER-Nov/2020- 28375-1

Dear Mr Kittisak Tipakornrojanakit,

The Research Ethics Committee has considered the above application recently submitted by you.

The Chair, acting under delegated authority has confirmed that, according to the information provided in your application, your project does not require ethical review.

Please note that:

- Approval to proceed with the study is granted providing that you do not carry out any research which concerns human participants, their tissue and/or their data.
- The Research Ethics Committee reserves the right to sample and review documentation relevant to the study.
- If during the course of the study, you would like to carry out research activities that concern a human participant, their tissue and/or their data, you must inform the Committee by submitting an appropriate Research Ethics Application. Research activity includes the recruitment of participants, undertaking consent procedures and collection of data. Breach of this requirement constitutes research misconduct and is a disciplinary offence.

Good luck with your research!

Kind regards,

[illegible]

Professor Hua Zhao

Chair of the College of Engineering, Design and Physical Sciences Research Ethics Committee

Brunel University London