جامعة الشهيد حمّه لخضر - الوادي
UNIVERSITY OF ELOUED

University of Echahid Hamma Lakhdar - El-Oued
Faculty of Exact Sciences - Computer Science department
Option: Distributed Systems and Artificial Intelligence

Master's Thesis

Detecting Ocular Ailments Using Artificial Intelligence

Presented by:
Mohammed Soufia
Seif-eddine Merkhoufi

Supervisor:
Dr. Messaoud Abbas

Co-supervisor:
Dr. Mohammed Mounir Bouhamed

Dr. Mohammed Abdelhamid Nedioui - Presedent
Dr. Samir Othmani - Examiner

Academic year 2023-2024

Abstract

Early diagnosis of ocular diseases is crucial for preventing vision loss and improving patient health. Optical Coherence Tomography (OCT) is a popular and effective imaging technique that provides detailed images of the eye retina and is used to diagnose various eye diseases. However, interpreting these images is challenging due to the complex anatomy of the eye and the fine and slight differences between healthy and diseased retina layers. This thesis uses Artificial Intelligence (AI) and deep learning to detect and analyse OCT images. On the other hand, detecting diseases from medical images using AI techniques faces several challenges, including image processing and the change in their operations before training. Therefore, we also built maintainable software for image processing. The framework is used to preprocess images before training machine learning models. This framework ensures reliability and adaptability, allowing for repeated and versatile use. We use both SOLID principles and Test-Driven Development to build this software. In this thesis, we contribute to the field of ocular diagnostics by collecting a dataset of OCT images, specifically from individuals of Arabian descent (Algerian). Furthermore, we train several machine learning and deep learning models for ocular diseases in OCT images. We used several models and techniques, including YOLOv8, transfer learning, Convolutional Neural Network (CNN), KNN and others. We also presented a framework for deploying our models for doctors, ensuring seamless integration into clinical practice and widespread utilisation.

Keywords: Optical Coherence Tomography (OCT), Maintainable Software, TDD, SOLID Principles, Deep Learning, Machine Learning.

Résumé

Un diagnostic précis et efficace des maladies oculaires est crucial pour prévenir la perte de vision et améliorer la santé des patients. La Tomographie par Cohérence Optique (TCO) est une technique d'imagerie courante et efficace qui fournit des images détaillées de la rétine et est utilisée pour diagnostiquer diverses maladies oculaires. Cependant, interpréter ces images est un défi en raison de l'anatomie complexe de l'œil et des différences fines et subtiles entre les couches rétiniennes saines et malades. Cette thèse utilise l'Intelligence Artificielle (IA) et l'apprentissage profond pour détecter et analyser les images TCO. D'un autre côté, les techniques d'IA rencontrent plusieurs défis lorsqu'il s'agit de détecter des maladies à partir d'images médicales, notamment le traitement des images et la modification de leurs opérations avant l'entraînement. Par conséquent, nous avons également construit un logiciel maintenable pour le traitement des images. Ce logiciel est utilisé pour prétraiter les images avant l'entraînement des modèles d'apprentissage automatique. Il assure la fiabilité et l'adaptabilité, permettant une utilisation répétée et polyvalente. Nous utilisons les principes SOLID et le Développement Dirigé par les Tests (DDT) pour construire ce logiciel. Dans cette thèse, nous contribuons au domaine du diagnostic oculaire en collectant un ensemble de données d'images TCO, spécifiquement de personnes d'origine arabe (algérienne). En outre, nous avons entraîné plusieurs modèles d'apprentissage automatique et d'apprentissage profond pour les maladies oculaires dans les images TCO. Nous avons utilisé plusieurs modèles et techniques, y compris YOLOv8, l'apprentissage par transfert, le réseau de neurones convolutifs (CNN), KNN et d'autres. Nous avons également présenté un cadre pour déployer nos modèles auprès des médecins, garantissant une intégration fluide dans la pratique clinique et une utilisation répandue.

Mots Clés: Tomographie par cohérence optique (OCT), Logiciel maintenable, TDD, Principes SOLID, Apprentissage profond, Apprentissage automatique.

ملخص

التشخيص الدقيق والفعال لأمراض العيون أمر بالغ الأهمية لمنع فقدان البصر وتحسين صحة المرضى. تعد تقنية التصوير المقطعي البصري OCT تقنية تصوير شائعة وفعالة توفر صورا مفصلة لشبكية العين وتستخدم لتشخيص مختلف أمراض العيون. ومع ذلك، فإن تفسير هذه الصور يشكل تحديا بسبب التشريح المعقد للعين والفروق الدقيقة والبسيطة بين طبقات الشبكية السليمة والمريضة. تستخدم هذه الأطروحة الذكاء الاصطناعي والتعلم العميق للكشف عن وتحليل صور OCT. من ناحية أخرى، تواجه تقنيات الذكاء الاصطناعي تحديات عديدة عند اكتشاف الأمراض من الصور الطبية، بما في ذلك معالجة الصور وتغيير عملياتها قبل التدريب. لذلك، قمنا أيضا ببناء برنامج صيانة لمعالجة الصور. يستخدم هذا الإطار لمعالجة الصور قبل تدريب نماذج التعلم الآلي. يضمن هذا الإطار الموثوقية وقابلية التكيف، مما يسمح بالاستخدام المتكرر والمتنوع. نستخدم مبادئ SOLID والتطوير بالاختبار أولاً لبناء هذا البرنامج (TDD). في هذه الأطروحة، نساهم في مجال تشخيص أمراض العيون من خلال جمع مجموعة بيانات من صور OCT، تحديدا من أفراد ذوي الأصول العربية (الجزائرية). بالإضافة إلى ذلك، قمنا بتدريب العديد من نماذج التعلم الآلي والتعلم العميق لأمراض العيون بواسطة صور OCT. استخدمنا العديد من النماذج والتقنيات، بما في ذلك YOLOv8، التعلم بالنقل، الشبكات العصبية التلافيفية CNN، KNN وغيرها. قدمنا أيضا إطارا لنشر نماذجنا للأطباء، مما يضمن التكامل السلس في الممارسة السريرية والاستخدام الواسع النطاق.

الكلمات المفتاحية: التصوير المقطعي البصري (OCT)، البرمجيات القابلة للصيانة، TDD، مبادئ SOLID، التعلم العميق، التعلم الآلي.

Contents

List of Figures

List of Tables

List of Listings

Introduction

Background

The ocular diseases affect millions worldwide, leading to vision problems and blindness if left untreated, which affects patients' quality of life and adds pressure on healthcare systems. Early detection helps prevent vision loss and manage these conditions effectively by interventions that can slow disease progression. Optical Coherence Tomography (OCT) is one technique that highlights the value of eye diagnoses and scanning and is used in the early detection of ailments. OCT provides high-resolution images of the retina's structure in great detail, allowing for the early detection of various ocular diseases.

Interpreting OCT images poses a human error challenge, as well as challenges such as variability in image quality due to factors like patient motion or lens artefacts. These issues can hide critical details and lead to incorrect diagnoses or delayed treatment. Moreover, the complexity of OCT images, which often require careful attention to the differences in tissue textures and structures, further increases the risk of misinterpretation. Additionally, the interpretation of OCT scans can be influenced by the interpreter's experience and familiarity with rare or unusual presentations.

Motivation

Artificial intelligence (AI) appears as a solution for more advanced and accurate detection methods to improve early diagnosis and treatment through better detection. Artificial intelligence has emerged as a powerful tool for improving early diagnosis and treatment in healthcare, particularly through machine learning and deep learning in medical imaging. As highlighted in [13], they used AI to segment retinal layers, and in [14] they used AI to classify multiple ocular diseases effectively. AI solutions could analyse medical images with remarkable accuracy, efficiency, and consistency, greatly enhancing diagnostic capabilities. They can rapidly process large volumes of OCT data, identifying patterns and anomalies that human eyes might miss. AI reduces the risks associated with manual analysis and human error by helping ophthalmologists with second opinions and identifying suspicious areas.

Preprocessing the images for training the machine learning model is a complex task, thus we aim to build a solution for image processing. Our focus is on maintainable software techniques to build a reusable framework. Moreover, by adhering to these practices, we ensure the longevity and scalability of our processing operations. Furthermore, maintaining a modular and reusable codebase allows us to integrate new features seamlessly, ensuring that our solution remains adaptable and responsive to evolving needs in image processing technology. Ultimately, these efforts support the creation of a sustainable and effective platform that can deliver accurate results and drive innovation in diagnostics or other fields that rely on image analysis.

Thesis Contribution

- A collected dataset of OCT retina images focusing on individuals of Arabian descent.
- Developing an image processing framework built on maintainable software techniques.
- Training machine-learning models for predicting ocular ailments in OCT images.
- Developing a deployment framework that facilitates using the model by doctors and medical staff.

Thesis Organisation

This thesis is organised into 5 chapters in addition to the general introduction and conclusion. Sections are structured as follows:

- **Chapter 1: Generalities on Eye and Optical Coherence Tomography** , explores the anatomy of the eye, focusing on the layers of the retina. It presents various diseases that can be captured with OCT images. And finishes by presenting the related works.
- **Chapter 2: Developing Maintainable Software** defines maintainable software and the main techniques for it.
- **Chapter 3: Image Processing** , presents the field of image processing and its importance and some helpful processing operations.
- **Chapter 4: Artificial Intelligence** presents the fundamentals of artificial intelligence by focusing on machine learning and deep learning and their algorithms and presents evaluation metrics and challenges in building ML models.
- **Chapter 5: Contributions, Tests and Experiments** presents the experiments and contributions, beginning with the gathered dataset, then the processing framework, then the ML model, and finishing with the deployment framework.

Chapter 1

Generalities on Eye and Optical Coherence Tomography

1.1 Introduction

The eye is one of the most important and complex organs. It is used to see around us and connect to the outer environment. Therefore, taking care of its health is vital. We should understand its components, how it works, and diagnoses tools to achieve this.

This chapter presents some generalities about eyes, and the use of Artificial Intelligence (AI) to detect ailments in medical images of eyes. We consider Optical Coherence Tomography (OCT) images in this work. Section 1.2 explains how humans see and describe the eye anatomy and the retina anatomy. Section 1.3 describes some fundamentals of OCT images in eye care. Section 1.4 presents some eye diseases that can be detected by reading OCT. Section 1.5 presents some related works regarding this topic aiming to present some practical solutions and research about OCT with machine learning, Deep Learning, and diagnostic and treatment techniques.

1.2 Human Eye Fundamentals

1.2.1 Functioning of Human Eye

Human vision is a complex sense. Humans see through the collaborative work of different parts of the eye [15]. Seeing starts with light entering the cornea, the transparent front layer of the eye, which bends the light due to its round shape. The light passes via the pupil to the lens, which works with the cornea to make the eye focus. When it arrives at the retina, the photoreceptor cells turn light into electrical signals and reach the brain through the optical nerve. Finally, the brain turns signals into images.

1.2.2 Eye Anatomy

The Human Eye has a spherical structure located on the frontal surface of the skull in a bone structure called the Orbit [15]. An adult eye has a sagittal diameter of approximately 24 millimeters, a transverse diameter of 24.5 to 25 millimeters, and a weight of around 7.5 grams. The eye is attached by six extraocular muscles that rotate and move the eye [15]. Figure 1.1 shows how the eyes are created inside the skull.

The Eyeball has three main layers:

- The outer layer is called the Outer Fibrous Layer.
- The middle layer is called the Middle Vascular Layer.
- The inner layer is called the Inner Neural Layer (Retina).

Figure 1.1: A Human Eye inside a Skull [1].

The Outer Fibrous contains the Sclera and the Cornea. The Sclera is the visible white part of the eye. It covers the whole eye except the front, which the Cornea covers. The Cornea continues the Sclera, a transparent layer that allows light to pass through. Together, they protect the eye [15].

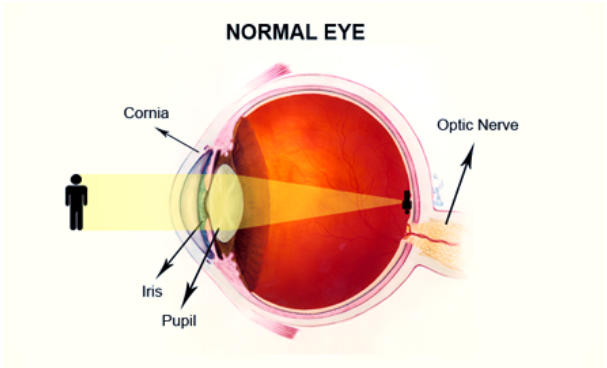

Figure 1.2: Functioning of Human Eye .

The Middle Vascular layer mainly contains the Choroid, Ciliary body, and Iris. The Choroid supplies the eye with blood that brings oxygen and nutrients. The Ciliary body contains smooth muscle fibres attached to the Lens, allowing the eye to focus on objects at different distances. The Iris, which is the coloured part of the eye, controls the amount of light that enters. The Pupil is a circular opening located at the centre of the Iris. The anterior part of the eye is divided into two chambers filled with liquid: the anterior chamber and the posterior chamber. The Lens works to focus on objects that are closer or further from us. The Vitreous Chamber is in the posterior part of the eye between the Lens and the Retina and contains the Vitreous Humour [15].

The Inner Neural Layer, or Retina, is formed by ten layers of different cell types. The Retina is a sensory membrane that covers the inner surface of the eye and processes light information. It is composed of two types of cells: Rods and Cons. The Retina is also divided into different areas: the Macula, with the Fovea in the middle, and the peripheral Retina. The Macula is in the central area of the Retina, providing central vision. The Fovea is a region in the center of the Macula that provides details about where the eye is focused, and both are formed of cons. The Peripheral Retina, populated mainly by Rods, is essential in providing peripheral vision and functioning effectively in low-light conditions [15]. Lastly, we have the optic nerve, which transfers signals to the brain [15].

The eye anatomy is shown in the Figure 1.3.

1.2.3 Retina Layers

According to reference [15], a Retina has ten layers.

- Inner Limiting Membrane is a boundary between the retina and the vitreous body.

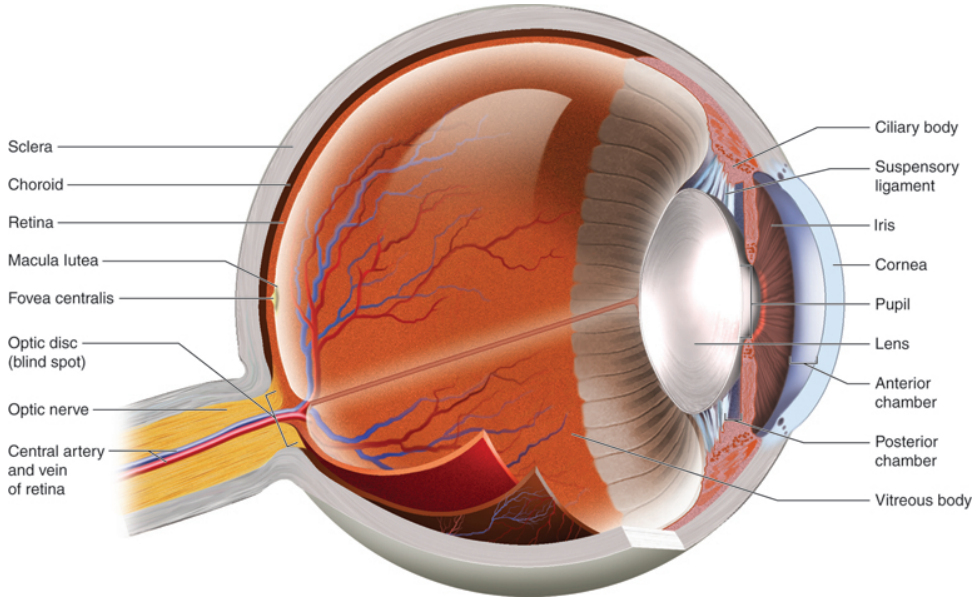

Figure 1.3: The anatomy of human eye [2].

- The nerve Fibre Layer is formed by the expansion of optic nerves.
- The ganglion Cell Layer contains ganglion cells and some displaced amacrine cells.
- The Inner Plexiform Layer is called the inner synaptic layer.
- The Inner Nuclear Layer contains cell bodies of various retinal cells.
- The Outer Plexiform Layer is known as the outer synaptic layer.
- The Outer Nuclear Layer contains the cell bodies of photoreceptor cells.
- The External Limiting Membrane that differentiates between segments of photoreceptor cells [15].
- Layer of Rods and Cones: Contains the photoreceptor cells.
- The Retinal Pigment Epithelium nourishes retinal visual cells and prevents reflection in the eyeball.

1.3 Optical Coherence Tomography

OCT is a modern ocular imaging process that makes it easy to visualize high-resolution cross-sectional images of the retina and other eye parts. They use light waves to capture images. They are used to diagnose various eye conditions since they provide detailed information about the retina layers, optic nerve, and other eye structures [16].

OCT images can show different layers based on light reflectance. Areas like the vitreous and outer nuclear layer allow light to pass through and look dark. Meanwhile, the retinal nerve fibre layer and RPE reflect brighter light.

Some brighter zones represent different parts of photoreceptor cells, which are crucial for light detection and signal transmission. The external limiting membrane, a highly reflective boundary; the foveal depression, crucial for sharp vision; and the RPE/Bruch's complex are vital for retinal health due to its combination of a nourishing cell layer and supportive tissue.

Retinal thickness and contour could vary depending on several factors including axial length, eye shape, [17]. (As shown in Figure 1.6)

Figure 1.6 presents a healthy retina captured with an OCT machine.

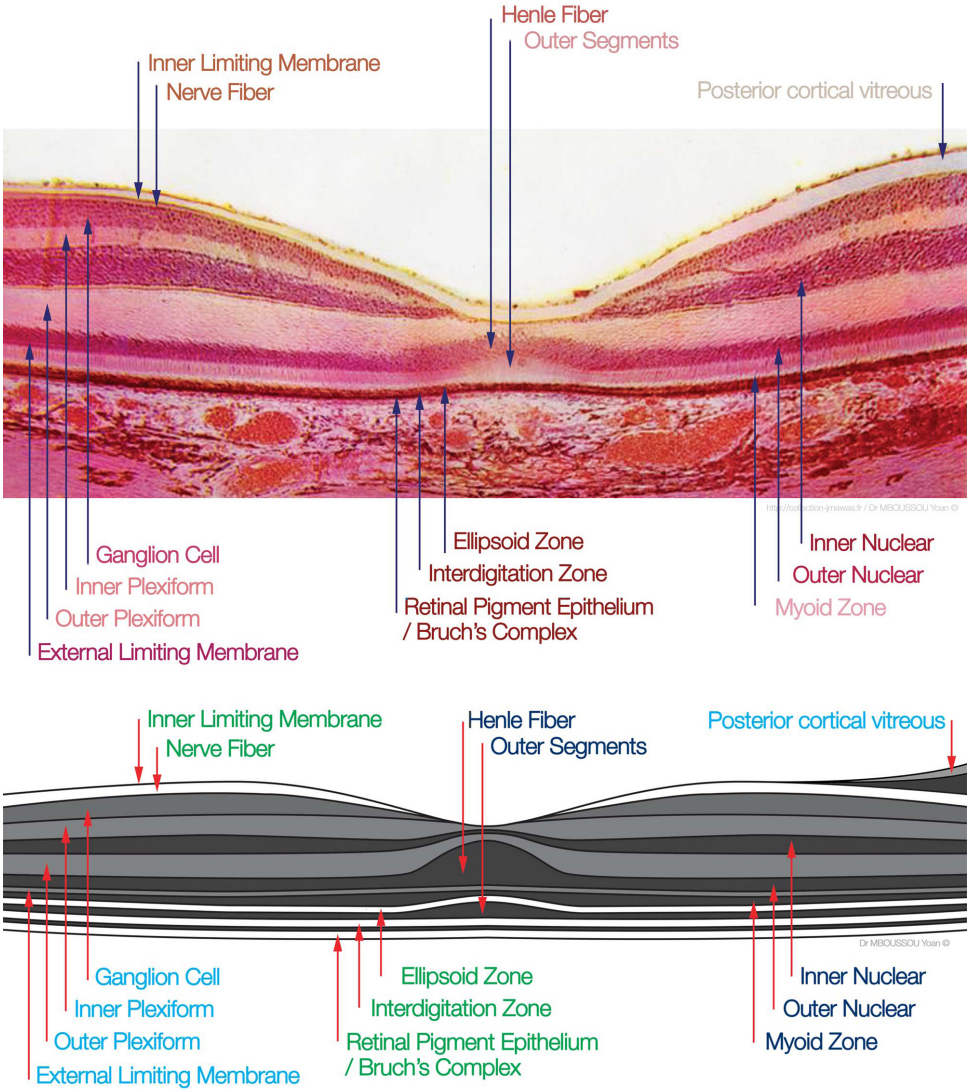

Figure 1.4: Retina layers [3].

Figure 1.5: OCT Machine

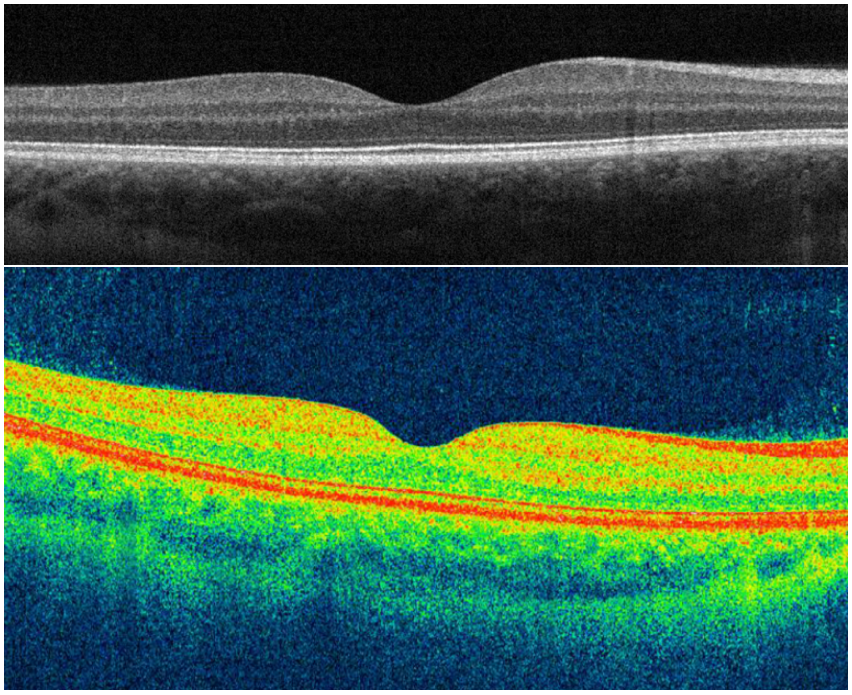

Figure 1.6: Healthy eye demonstrated by OCT imaging.

1.4 Ocular Diseases with OCT

According to references [16] and [18], several ocular diseases could be captured from OCT images, which helps early detection and treatment. We cite some of these diseases:

1.4.1 Age-related Macular Degeneration (AMD)

Age-related macular degeneration (AMD) is a progressive eye condition that affects the macula, the central part of the retina responsible for sharp central vision. It typically occurs in older adults and can lead to a gradual loss of central vision, making tasks like reading and driving difficult. There are two main types: dry AMD, as demonstrated in figure 1.8, which involves the accumulation of drusen (yellow deposits) under the retina, as shown in figure1.9, and wet AMD, which involves abnormal blood vessel growth that can leak fluid and blood into the retina as shown in figure 1.7. While there is no treatment for AMD, medications or laser therapy may help slow its progression and preserve remaining vision [16] [18].

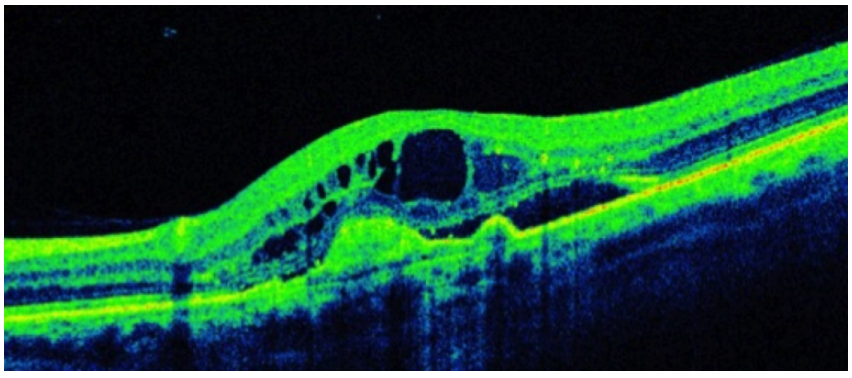

Figure 1.7: Wet AMD demonstrated by OCT imaging (1).

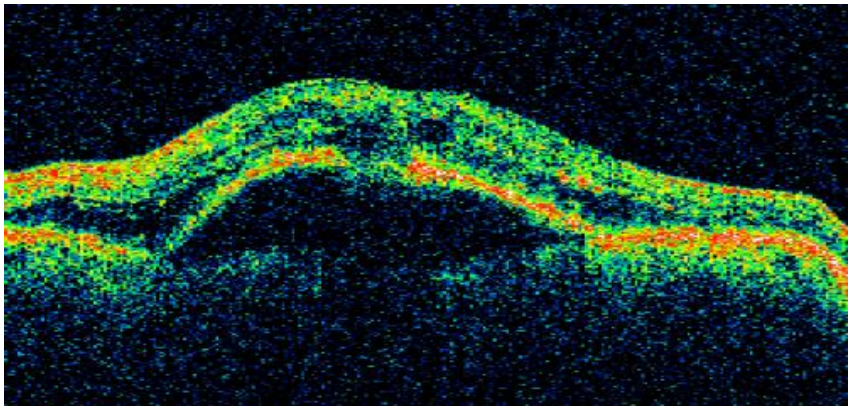

Figure 1.8: Wet AMD demonstrated by OCT imaging (2).

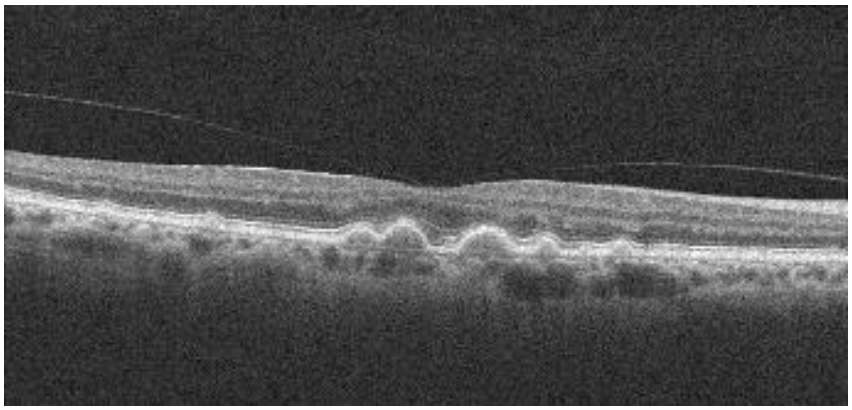

Figure 1.9: Drusen in AMD demonstrated by OCT imaging.

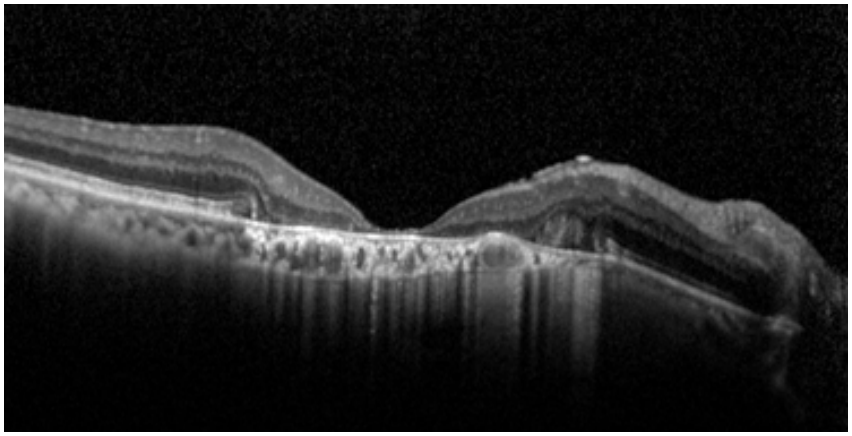

Figure 1.10: AMD demonstrated by OCT imaging.

1.4.2 Retinal Vein Occlusion (RVO)

Retinal vein occlusion occurs when a vein in the retina becomes blocked, usually due to a blood clot. This blockage can lead to sudden, painless vision loss in one eye. Symptoms may include blurred vision, distorted vision, or a sudden floater increase. Treatment options depend on the severity and location of the occlusion. They may include medications to reduce swelling, laser treatment to improve blood flow, or injections to dissolve blood clots and prevent further complications [16], Figures 1.11,1.12 and 1.13 demonstrates RVO [18].

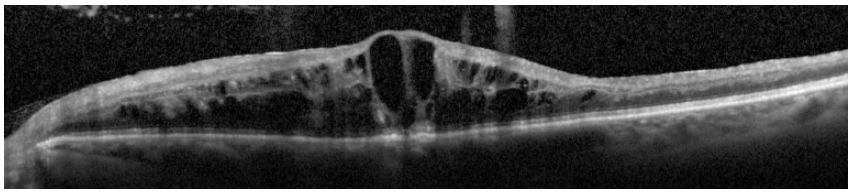

Figure 1.11: RVO demonstrated by OCT imaging (1).

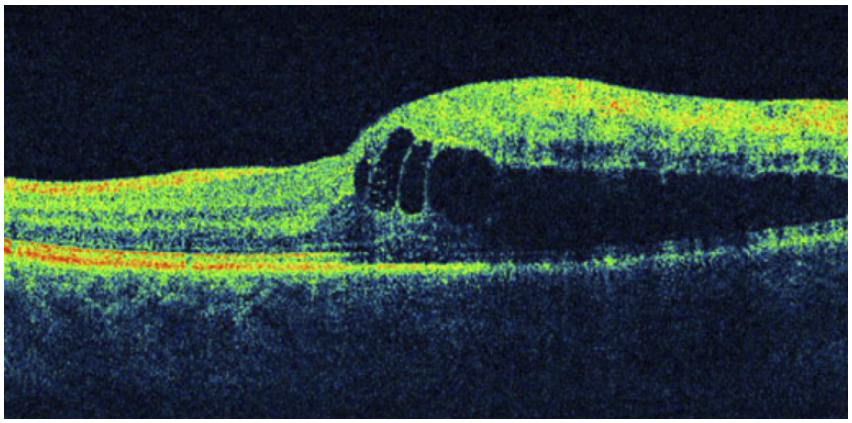

Figure 1.12: RVO demonstrated by OCT imaging (2).

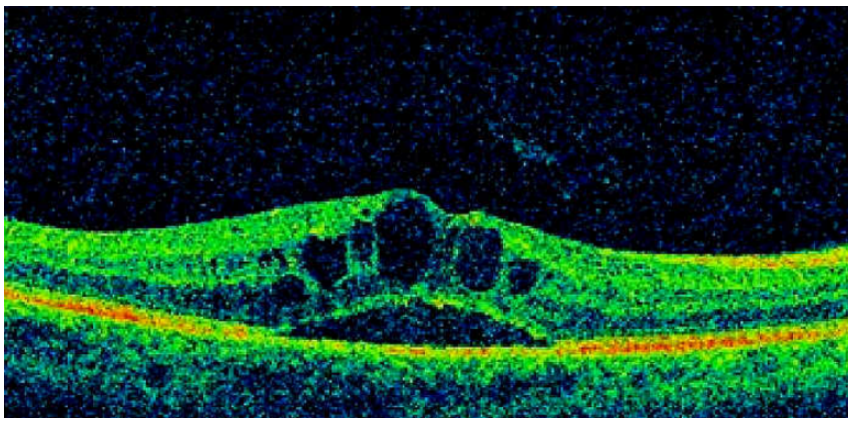

Figure 1.13: RVO demonstrated by OCT imaging (3).

1.4.3 Central Serous Chorioretinopathy (CSCR)

CSCR is a condition where fluid accumulates under the retina, causing a detachment of the central retina (macula). It typically affects one eye and can lead to blurred or distorted central vision and seeing straight lines as wavy. CSCR is often associated with stress or steroid medication use, and it usually resolves

on its own within a few months without treatment. However, if symptoms persist or worsen, treatments like laser therapy or medication may be considered to expedite recovery [16]. CSCR is demonstrated in Figure 1.14).

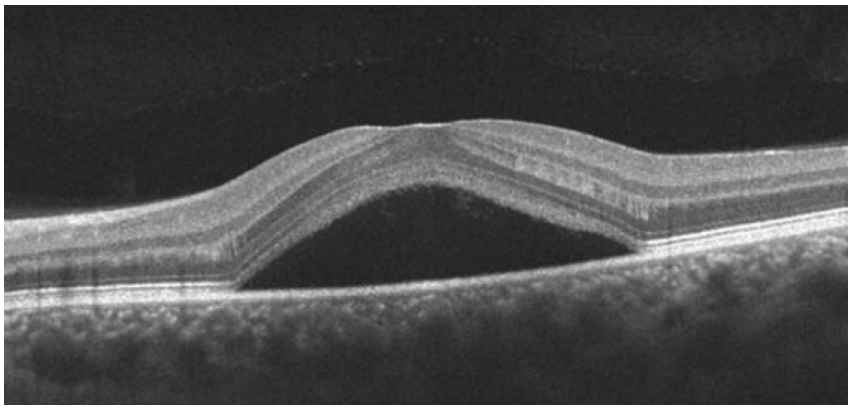

Figure 1.14: CSCR demonstrated by OCT imaging.

With CSCR, there can often be a component of pigment epithelial detachment (PED) inside the area of serous detachment. These PEDs can be large, as in Figure 1.15).

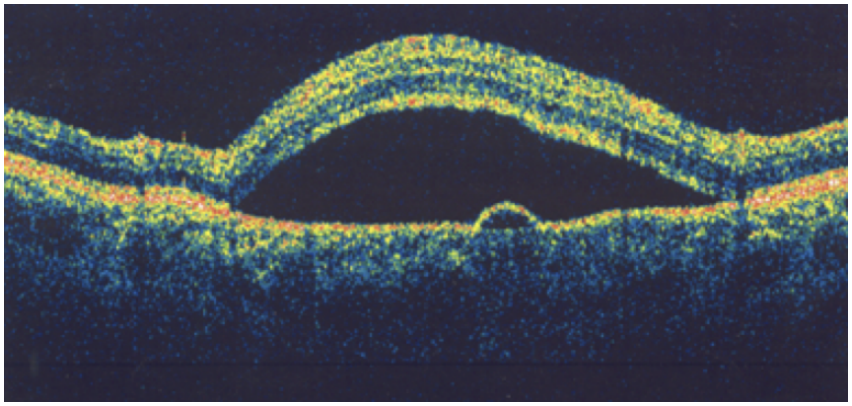

Figure 1.15: PED in CSCR demonstrated by OCT imaging.

This example of CSCR displays a very thick choroid [18] 1.16).

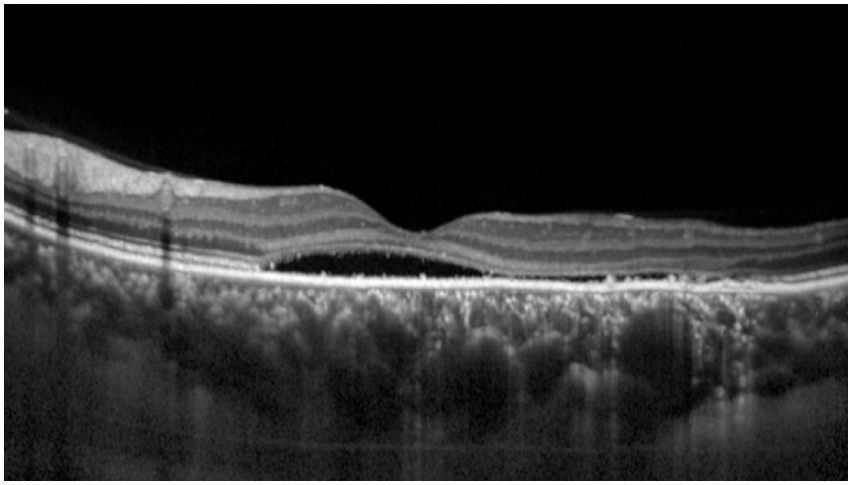

Figure 1.16: Thick choroid in CSCR demonstrated by OCT imaging.

1.4.4 Epiretinal Membrane (ERM)

An epiretinal membrane is a thin scar tissue layer forming on the retina's surface. It can cause mild to moderate distortion or blurring of central vision. Symptoms may include seeing wavy or distorted lines and sometimes decreased visual acuity. Treatment options vary depending on the severity of symptoms and may include observation, medications, or surgery to remove the membrane and restore clearer vision. A dense ERM in figure 1.17 leads to inner retinal wrinkling and distortion of the foveal contour [16].

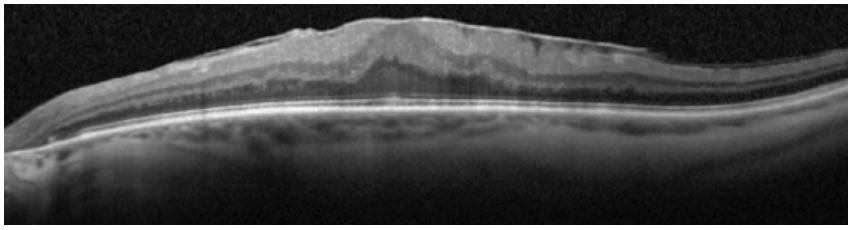

Figure 1.17: Dense ERM demonstrated by OCT imaging.

A light to moderate ERM is shown in figure 1.18) [18].

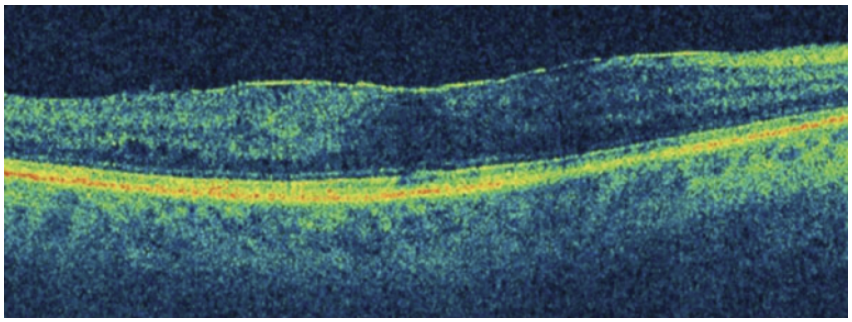

Figure 1.18: Mild to ERM demonstrated by OCT imaging.

1.4.5 Macular Hole (MH)

A macular hole is a small break in the macula, the central part of the retina responsible for sharp, detailed vision. It can cause blurred and distorted central vision. Macular holes often result from age-related changes but can also be due to eye injuries or other conditions. Treatment typically involves surgery to close the hole and restore vision [16].(see Figure 1.19)

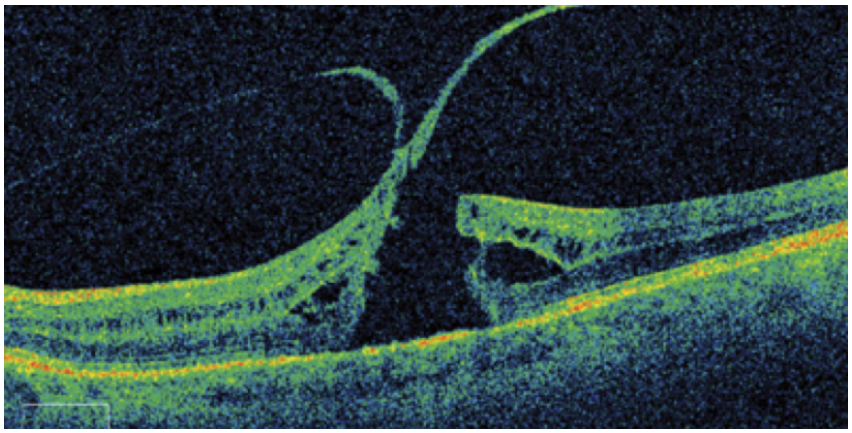

Figure 1.19: MH demonstrated by OCT imaging.

Here, the posterior hyaloid has separated, leaving a central operculum and a full-thickness defect [18] (see Figure 1.20).

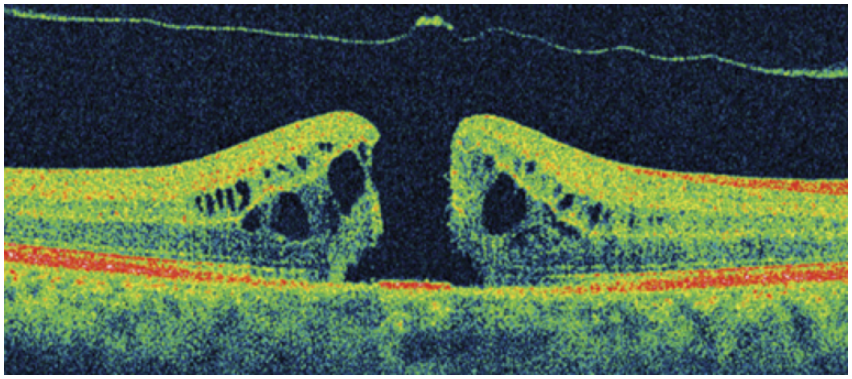

Figure 1.20: MH with separated posterior hyaloid demonstrated by OCT imaging.

1.4.6 Retinitis Pigmentosa (RP)

Retinitis pigmentosa is a genetic disorder that affects the retina's ability to respond to light. It typically causes a gradual loss of peripheral vision, leading to tunnel vision over time. Symptoms may include difficulty seeing in low light, decreased night vision, and central vision loss in later stages. There is no treatment for retinitis pigmentosa, but treatments can help manage symptoms and slow its progression [16].(see Figure 1.21).

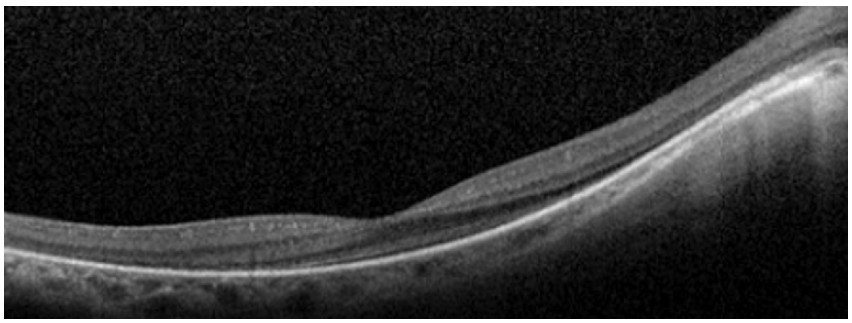

Figure 1.21: RP demonstrated by OCT imaging.

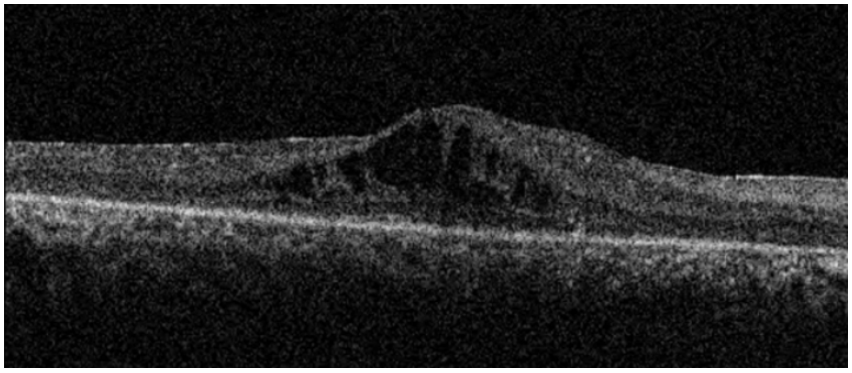

Figure 1.22: RP demonstrated by OCT imaging.

1.4.7 Retinal Detachment

Retinal detachment occurs when the retina light-sensitive layer at the back of the eye, separates from its underlying tissue. This condition can lead to sudden symptoms like flashes of light, floaters, and a shadow or curtain over part of the visual field. It requires prompt medical treatment, often surgery, to reattach the retina and prevent permanent vision loss [16]. (see Figure 1.23).

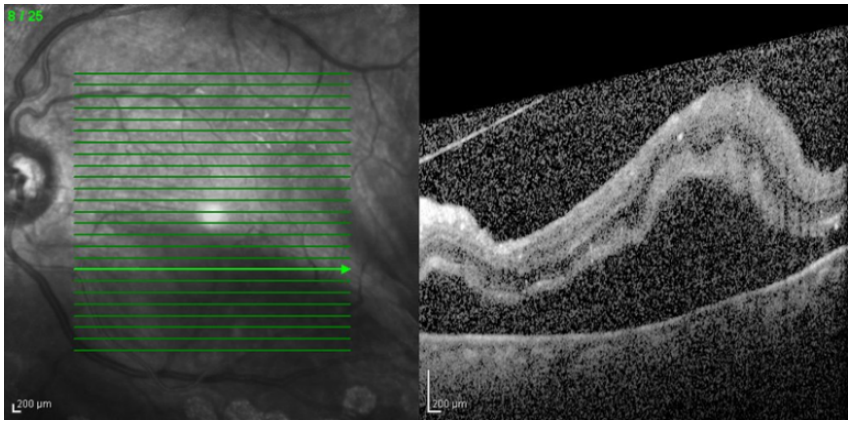

Figure 1.23: Retinal detachment demonstrated by OCT imaging.

1.5 Related Works

Researchers use different techniques to recognise diseases in OCT images, such as inspecting and examining the images visually and identifying abnormalities and diseases. They also compare the different OCT images of the patient to find differences and irregularities. These techniques are based on skill and experience, making non-experienced doctors misinterpret images, which is dangerous to patient health.

Recent medical studies highlight the power of deep learning for analyzing OCT images. Research has demonstrated the success of analyzing retinal images for diseases like DME and AMD. Machine learning and deep learning algorithms are efficient since they enable rapid analysis of vast datasets and facilitate timely detection through pattern recognition to identify diseases.

In 2017, research [19] focused on classifying DME and normal retinas using a pre-trained CNN model, using VGG16.

In 2017, research [20] focused on classifying normal and AMD images using deep neural networks. It used a large dataset of nearly 100.000 images, with an accuracy of 0.8763 on the image level.

In 2018, research [21] focuses on classifying AMD, DME or RVO using deep learning algorithms.

In 2019, Research [22] applied deep learning with several network models to detect AMD and DME. They use transfer learning and achieve the best result with CliqueNet, with an accuracy of 0.98.

In 2019, research [23] uses deep learning to segment retina layers by combining fully convolutional networks (FCN) with Gaussian Processes for post-processing.

In 2020, research [13] aims to segment retinal layers using deep learning. It has used the improved Xception65 network for feature extraction and achieved a mean intersection over union (IoU) of 0.9041 and a sensitivity (Se) of 0.9215.

In 2022, research [14] classified four categories: Diabetic Macular Edema (DME), Choroidal Neovascularization (CNV), Drusen, and Normal using deep learning CNN and used 84000 OCT images for this task.

Table 1.1: Summary of Datasets, Models, Diseases and Applications, and Accuracy

Ref	Dataset	Model Type	Diseases and Applications	Accuracy
[19]		CNN with VGG16 for feature extraction	DME, Healthy Retina	0.875
[20]	100.000 images		Healthy Retina, AMD	0.8763
[21]	1200 OCT scans, from Vienna Reading Center database	CNN	AMD, DME, RVO	accuracy of detecting SRF: 0.92
[22]	public dataset of 3231 OCT images provided by Srinivasan et al., obtained in Universities of Duke, Harvard and Michigan	CliqueNet	AMD, DME	0.98
[23]	OCT images provided by University of Miami	FCN	Retina layers segmentation	mean unsigned error of 1.06
[13]		Improved Xception65	Retinal layers segmentation	
[14]	84000 oct images	CNN	DME, CNV, Drusen, and Healthy Retina	0.9990

1.6 Conclusion

In this chapter, we presented the structure and function of the human eye. We explained how the eye perceives the outer environment by entering the light and turning it into electrical signals that travel to the brain using the optical nerve. We also presented the eye retina layers, their shape and appearance, and the importance of OCT images in diagnosing the retina layers. We showed some diseases that can be detected using OCT images, which improves the treatment and prevents the eye from illness or vision loss. Finally, we highlighted some related works.

Chapter 2

Developing Maintainable Software

2.1 Introduction

The statistics show that around 60% to 70% of the total software development cost is spent on maintaining traditional software with legacy code [24]? It also grows over time. Therefore, Investing in maintainability could reduce the cost of savings.

Maintainable software is a famous and essential field of software engineering. By designing maintainable software products, we ensure the sustainability and flexibility of software, making the development process much faster and at a lower cost. It is also important to reuse developed modules.

First, Section 2.2 defines maintainable software and essential techniques to develop it. Section 2.3 presents some famous code quality metrics. We explain design patterns in section 2.4. Lastly, 2.5, we explain how to use automated tests and Test-Driven Development (TDD) to create maintainable software.

2.2 Techniques of Developing Maintainable Software

2.2.1 Definition

Maintainable software is easy to understand, modify, fix, and extend throughout its lifecycle [25]. Several techniques are used to develop maintainable software including clean code, SOLID principles, code quality metrics, design patterns, automated tests, and TDD.

2.2.2 Clean Code

Clean code is a code that is easy to maintain. It follows principles and best practices that promote readability, simplicity, and efficiency and allow others to comprehend and work with it [25].

Key considerations for clean code include:

- names should be meaningful.
- functions should be small.
- form should be well formatted.

2.2.3 SOLID Principles

SOLID is an acronym for five design principles in software engineering meant to make object-oriented architectures more understandable, flexible, scalable, and maintainable [26]. Those principles were not entirely new concepts. However, the American software engineer and instructor Robert C. Martin promoted them 2.1. The SOLID principles are:

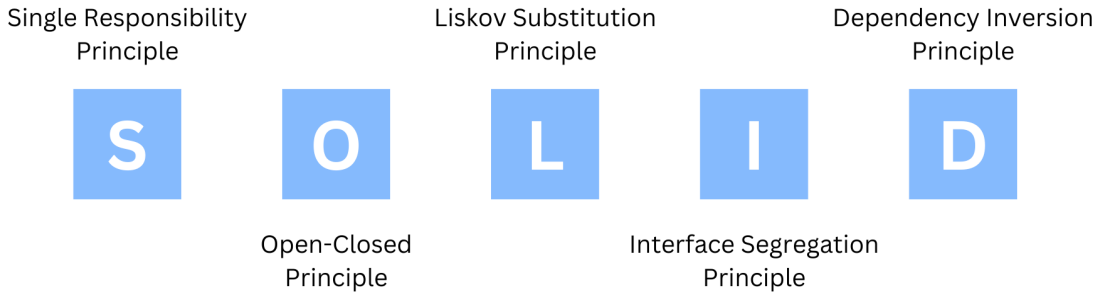

Figure 2.1: SOLID Principles for Maintainable Software Design.

- **Single Responsibility Principle (SRP):**

  A class should have only one responsibility, which gives it only one reason to change. This makes code easier to understand, maintain, and test [26].

  Example :

  In figure 2.2, there is an *Invoice* class that has four methods: *AddInvoice()*, *DeleteInvoice()*, *GenerateReport()*, and *EmailReport()*. *Invoice* class violates the SRP by having multiple methods with different reasons to change [27].

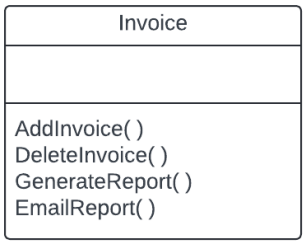

Figure 2.2: Class diagram showing violation of the SRP.

We should group methods with similar functionality in single classes and separate the different ones To respect the SRP. We will group *AddInvoice()* and *DeleteInvoice()* into a single class. *Invoice*. *GenerateReport()* will be in the *Report* class, and *EmailReport()* will be a part of the *Email* class. Figure 2.3 shows the new class diagram. Which makes the code smaller and more manageable.

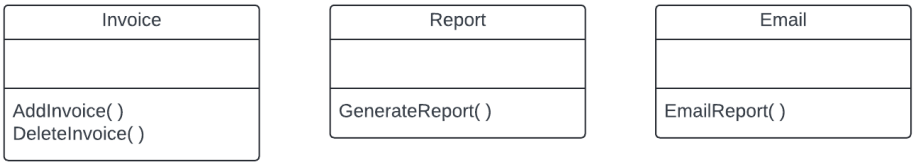

Figure 2.3: Class diagram showing respect of the SRP.

- **Open/Closed Principle (OCP):**

  The software entities (classes, modules, functions. etc.) should be opened for extension but closed for modification. In other words, the behaviour of a component should be easily extendable without needing to change its source [26].

  Example:

  In this example, the *AreaCalculator* class violates the OCP because it needs to be modified every time a new shape is introduced. When adding a new shape (e.g., triangle), the *AreaCalculator* class must be modified to adapt to the new shape. Figure 2.4 shows the example class diagram [28].

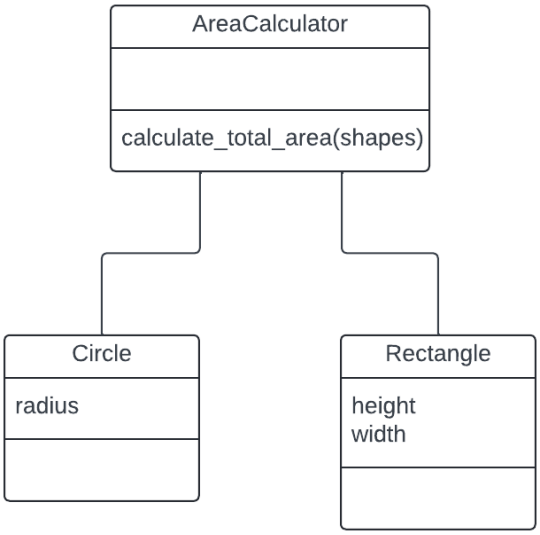

Figure 2.4: Class diagram showing violation of the OCP.

The *AreaCalculator* should iterate over the shapes and call each shape's *area()* method. Figure 2.5 demonstrates the relation between classes.

There's no need for if-else conditions based on the type of shape, so the *AreaCalculator* class is closed for modification when new shapes are introduced which respects the OCP.

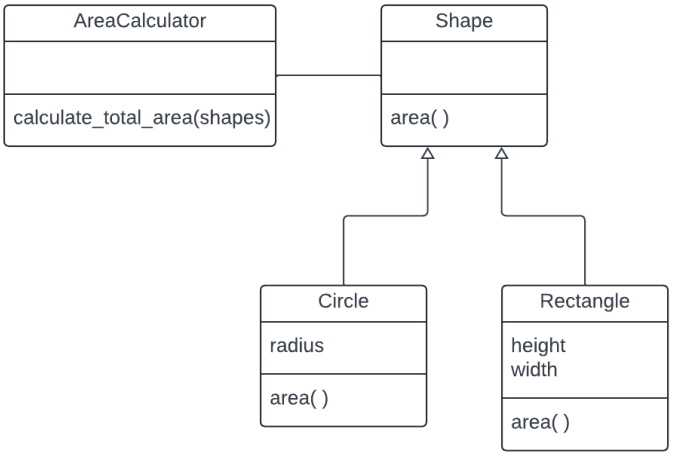

Figure 2.5: Class diagram showing respect of the OCP.

- **Liskov Substitution Principle (LSP):**

  The objects of a superclass should be replaceable with objects of its subclasses without affecting the correctness of the program [26].

  Example:

  The figure 2.6, present *CarInterface* with a couple of methods that all cars should be able to fulfill: turning on the engine and accelerating forward. We violate the LSP by introducing a car without an engine, resulting change the behaviour of our program [29].

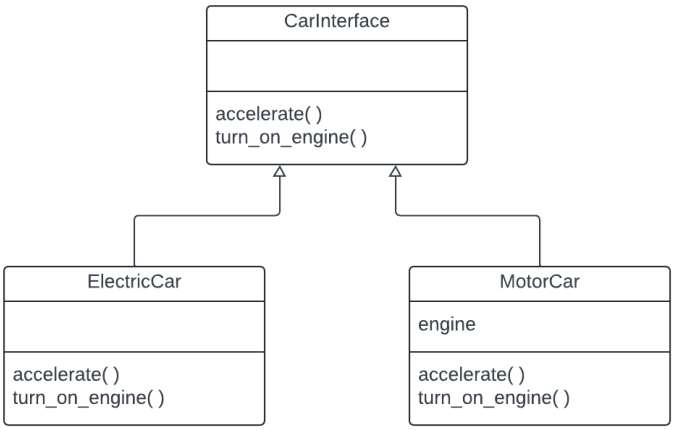

Figure 2.6: Class diagram showing violation of the LSP.

To follow the LSP, code should be organized with interfaces reflecting specific car behaviour. In the corrected version shown in 2.7, *MotorCarsInterface* and *ElectricCarsInterface* are separated, preventing unexpected behaviour when replacing types.

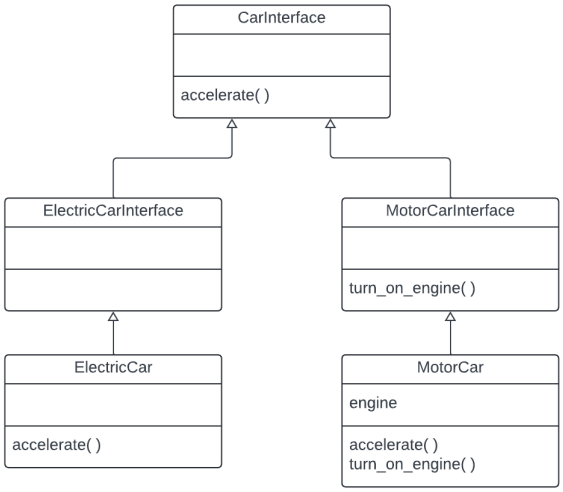

Figure 2.7: Class diagram showing respect of the LSP.

- **Interface Segregation Principle (ISP):**

  Clients should not be forced to use interfaces with methods that do not use. It suggests breaking large interfaces into smaller ones that fit the specific needs of each class. This keeps things simple and avoids unnecessary dependencies [26].

  Example:

  In this example 2.8, The *PrinterInterface* provides abstract methods for subclasses to implement. *OldPrinter* inherits from Printer and must implement all its methods. However, it doesn't use *fax* and *scan* methods, which violates the ISP [30].

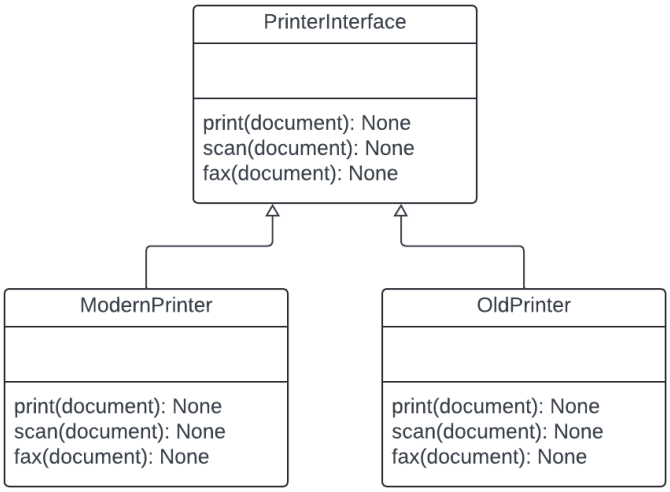

Figure 2.8: Class diagram showing violation of the ISP.

We have to separate *PrinterInterface* into specific interfaces and create concrete classes that inheritis from multiple interfaces as needed to respect the OCP as figure 2.9 shows. *OldPrinter* inherits the *PrinterInterface* and avoids unused *fax* and *scan* methods. *ModernPrinter* inherits all interfaces. This allows for creating different machines with different functionalities, making the design more flexible and extensible.

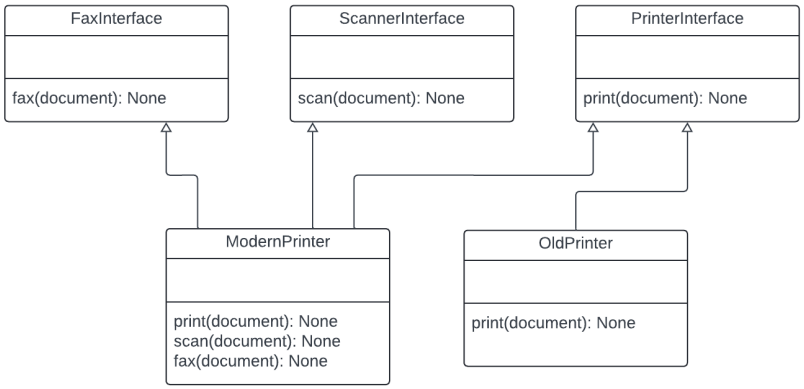

Figure 2.9: Class diagram showing respect of the ISP.

- **Dependency Inversion Principle (DIP):**

  High-level modules should not depend on low-level modules, both should depend on abstractions. This promotes decoupling and easier substitution of components, leading to more maintainable code [26].

  Example:

  In the example 2.10, *UserService* depends on the concrete implementation of *MySQLDatabase*, which violates DIP since the high-level class *UserService* is directly dependent on a low-level class. If we want to switch to a different database system, we must modify the *UserService* class [31].

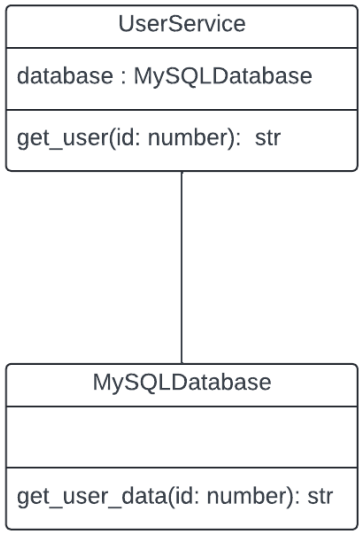

Figure 2.10: Class diagram showing violation of the DIP.

The class *UserService* should depend on abstraction. We should create a *DatabaseInterface* as an abstraction and create concrete implementations *MySQLDatabase* and *PostgreSQLDatabase*. This way, the *UserService* class depends on the Database abstraction, respecting the Dependency Inversion Principle 2.11.

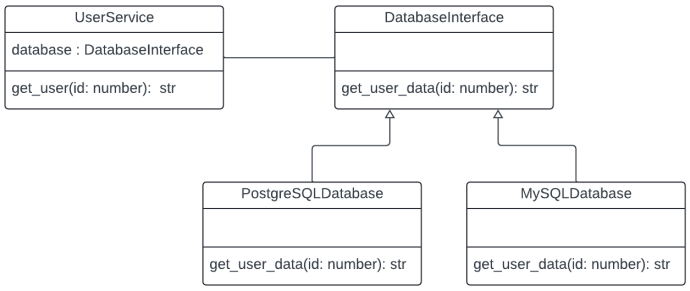

Figure 2.11: Class diagram showing respect of the DIP.

2.3 Code Quality Metrics

2.3.1 Definition

Code quality metrics are a collection of measures developers use to evaluate the quality and maintainability of code and estimate its complexity.

2.3.2 Frequently used Code Quality Metrics

Readability

Code readability is an index that differentiates between different developers and teams. Still, it generally follows the best practices like syntax, line length, consistency, and structure, making the code readable for all developers [25].

Code Cohesion

Code cohesion is the degree of the relation between components that serve a common task. When the components are strongly related and complete each other, it means they are highly cohesive [26].

Code Coupling

Code coupling is the degree of interdependence between components of software. High coupling means changing part of the code requires modification of another part, which makes the code more complex [26].

Code Coverage

Code coverage or test coverage is a percentage of how much code the test suite executes. In other words, how much of the source code is tested [32].

Lines of Code

Lines of Code is a simple metric of counting the number of lines in the codebase. This measure is not very accurate in measuring the code quality because the shorter code may be more complex or hard to maintain, but it still manages the growth and complexity [25].

2.4 Design Patterns

2.4.1 Definition of Design Patterns

Design patterns are typical solutions to common problems in software design. We can customize each pattern to solve a particular design problem. By using them, developers can make their software better organized, easier to change, and to understand [33].

2.4.2 Used Design patterns

Facade Design Pattern

A facade pattern is a structural design pattern that provides a simplified interface to a complex subsystem. It delegates tasks of various classes within the subsystem and hides its internal dependencies [33].

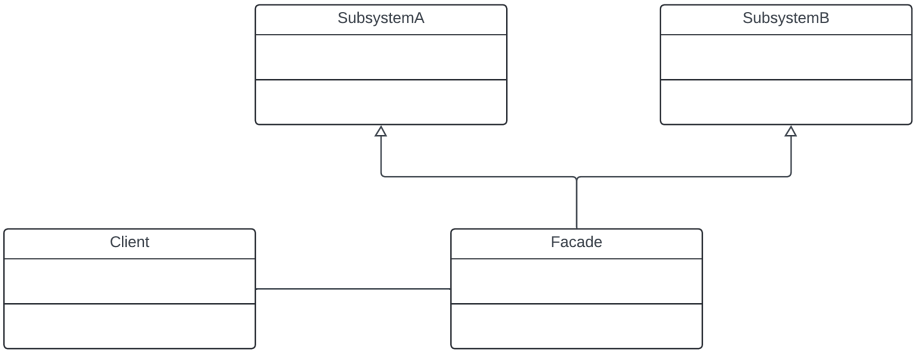

Figure 2.12: Facade Design Pattern Class Diagram.

Domain-Event Design Pattern

The domain event pattern is a behavioural pattern that facilitates loosely coupled communication between components by using events to represent significant changes within the application's domain. Events are raised by one component and handled by others, allowing for decoupled processing and asynchronous execution [4].

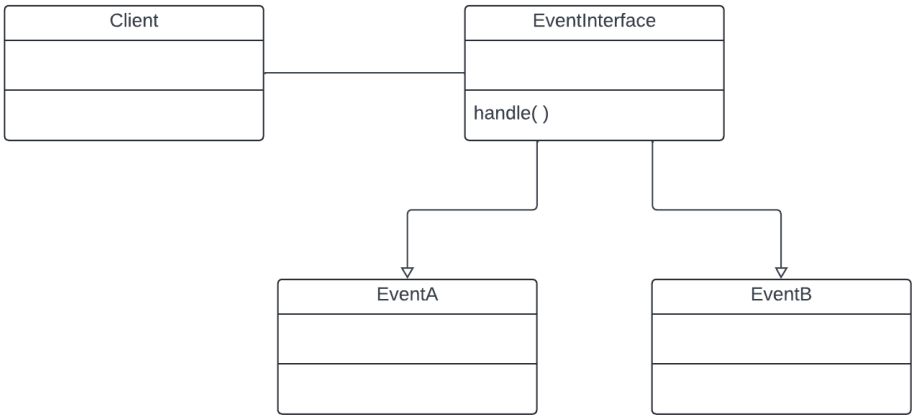

Figure 2.13: Domain-Event Design Pattern Class Diagram.

Composition Root Design Pattern

Composition root is a structural design pattern that is commonly used in the context of dependency injection. It is where all the application modules and components are initialized and usually located in the main function. It isolates the dependency setup in a single place, which makes code cleaner by separating the configuration from application logic [4].

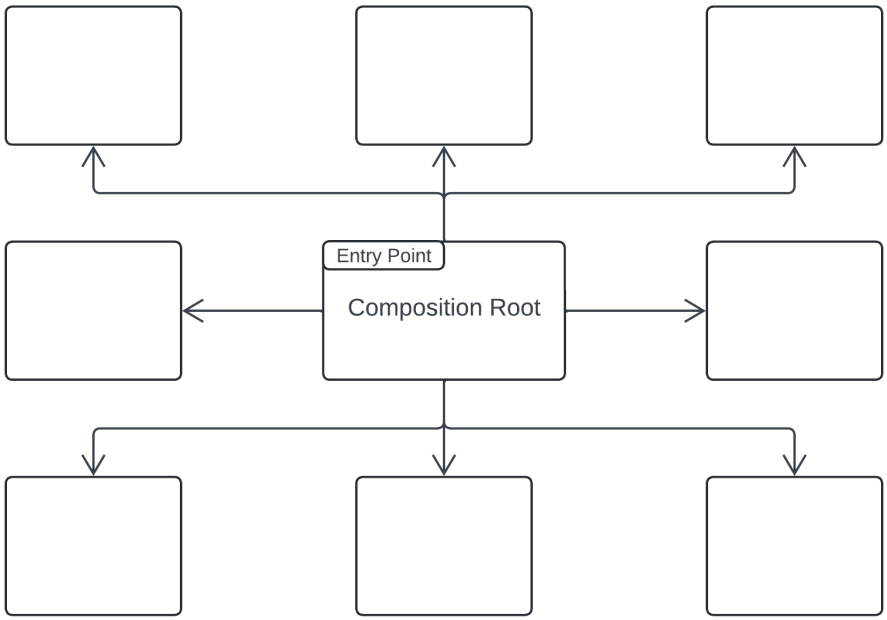

Figure 2.14: Composition Root Pattern Class Diagram [4].

2.5 Test-Driven Development

2.5.1 Automated tests

Automated tests are scripts or programs that minimise the human effort in the validation of the system functionalities by simulating user interactions through predefined steps and expected outcomes [32].

Example: an automated test to test the behaviour of a calculator on multiplication 2.1.

```
def test_calculator_multiplication():
    calculator = Calculator()
    result = calculator.multiply(12, 26)
    assert result == 312
```

Listings 2.1: Automated Test to Multiplication Behavior.

2.5.2 Understanding Test-Driven Development

Test Driven Development (TDD) is a technique used to develop software that focuses on creating automated unit tests before developing the production code [32]. Kent Beck created the approach in the late 1990s as part of eXtreme Programming (XP) and the Agile Manifesto. It is an iterative approach of combining unit test creation, programming, and refactoring.

2.5.3 Software Testing in TDD

Software testing in Test-Driven Development (TDD) refers to the practice of writing tests before writing the actual code [32]. In TDD, we should follow the pattern of Red-Green-Refactor 2.15:

Red: Write a failing test for the next functionality we want to add.

Green: Write the simplest code that passes the test.

Refactor: Refactor the new and the old code to structure it well.

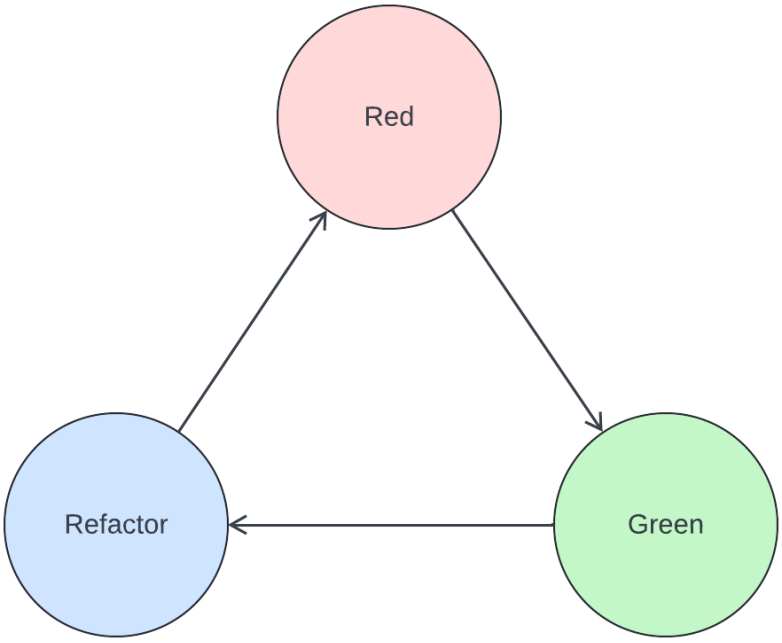

Figure 2.15: Test-Driven Development (TDD) Cycle.

In TDD, developers create small test cases for every feature based on their initial understanding. Developers gain clarity on the requirements and design of their code by writing tests first, resulting in cleaner and more maintainable software [32]. These tests are typically automated and can be run frequently to provide rapid feedback on the codebase health.

2.5.4 Software Test Doubles

A test double is a simulated object that imitates the behaviour of a real object in a controlled way. Doubles are often used in testing to isolate a particular module or component and to verify that it behaves as expected. This helps testers focus on the specific functionality without the complexities of external interactions [32].

Doubles tests are categorized into several types shown in the figure 2.16:

We will apply the example provided in 2.2 as a context for testing doubles.

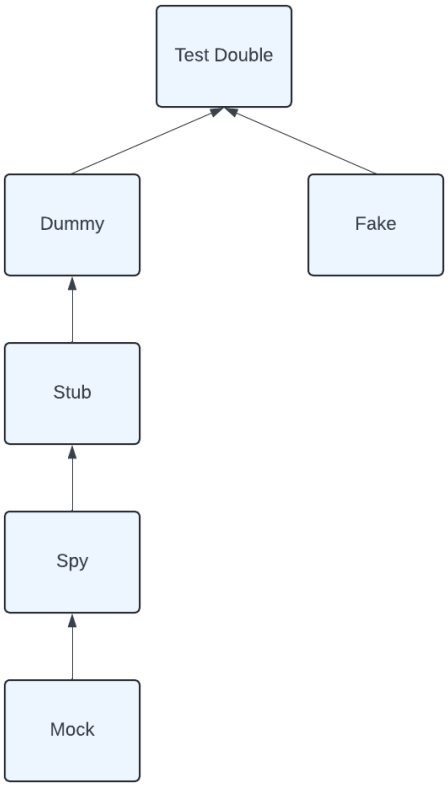

Figure 2.16: Types of Test Doubles Objects.

```python
from ABC import ABCMeta, abstractmethod

class PaymentGatewayInterface(metaclass=ABCMeta):
    @abstractmethod
    def process_payment(self, amount: int) -> bool:
        """abstract method"""

class PaymentProcessor(PaymentProcessorInterface):
    def __init__(self, gateway: PaymentGatewayInterface) -> None:
        self.gateway: PaymentGatewayInterface = gateway

    def make_payment(self, amount: int) -> bool:
        return self.gateway.process_payment(amount)
```

Listings 2.2: Payment Gateway Interface and Processor Implementation

Dummies

Dummy objects are doubles used as placeholders in a test. They are frequently used to satisfy method signatures or other requirements of the code being tested. However, the system test does not use or exercise them [32].

Example:

In this example, we will verify that the processor has an attribute named *gateway* 2.3 and utilise a dummy 2.4 object to complete this test.

```python
def test_payment_processor_should_have_gateway_attribute():
    dummy_gateway = DummyPaymentGateway()
    processor = PaymentProcessor(dummy_gateway)
```

```
    assert hasattr(processor, 'gateway')
```

Listings 2.3: Unit Test for Payment Processor with Dummy Gateway

```
class DummyPaymentGateway(PaymentGatewayInterface):
    def process_payment(self, amount: int) -> bool:
        pass
```

Listings 2.4: Dummy Payment Gateway Implementation

Stubs

A stub is a test doubles similar to the dummy but it is programmed to return a fixed, predetermined response at method calls [32].

Example:

In this example, we ensure that when making a payment, it returns the status of the operation 2.5 and utilises a stub 2.6 object to complete this test.

```
def test_make_payment_with_stub(self):
    gateway_stub = PaymentGatewayStub()
    processor = PaymentProcessor(gateway_stub)
    result = processor.make_payment()
    assert result == True
```

Listings 2.5: Unit Test for Payment Processor with Dummy Gateway

```
class StubPaymentGateway(PaymentGatewayInterface):
    def process_payment(self, amount: int) -> bool:
        return True
```

Listings 2.6: Stub Payment Gateway Implementation

Spyies

A spy object is a type of double that records information about its usage. It allows us to confirm that a function was called as expected and verifies the arguments passed to it [32].

Example:

In this example, we want to assert that when we call *make_payment* in the processor the *process_payment* function of Gateway is called 2.7, and we will use Spy 2.8 to achieve that.

```
def test_make_payment_processes_payment_with_payment_gateway():
    spy_gateway = SpyPaymentGateway()
    processor = PaymentProcessor(spy_gateway)
    transaction_result = processor.make_payment(100)
    assert spy_gateway.process_payment_call_count == 1
```

Listings 2.7: Unit Test for Payment Processor with Spy Gateway

```
class SpyPaymentGateway(PaymentGatewayInterface):
    def __init__(self) -> None:
        self.process_payment_call_count = 0

    def process_payment(self, amount: int) -> bool:
        self.process_payment_call_count += 1
        return True
```

Listings 2.8: Spy Payment Gateway Implementation

Mocks

Mock doubles are used to verify the behaviour of the tested code. They verify interactions between the unit under test and its dependencies [32]. Mocks can be programmed with specific expectations of how they should be used.

Example:

This example aims to verify if the *make_payment* method returns a transaction status, indicating whether the payment went through 2.9. We will simulate a payment *gateway* using *MockPaymentGateway* in the code 2.10, ensuring our payment processor communicates correctly.

```
def test_make_transaction_returns_transaction_result():
    mock_gateway = MockPaymentGateway()
    processor = PaymentProcessor(mock_gateway)
    transaction_result = processor.make_payment(100)
    assert transaction_result == True
```

Listings 2.9: Unit Test for Payment Processor with Mock Gateway

```
class MockPaymentGateway(PaymentGatewayInterface):
    def process_payment(self, amount: int) -> bool:
        if amount < 0:
            raise ValueError("Invalid amount value")

        if not sender_account.has_enough_balance():
            raise ValueError("Sender account does not have enough balance.")

        sender_account.withdraw(amount)
        receiver_account.deposit(amount)
        return True
```

Listings 2.10: Mock Payment Gateway Implementation

Fakes

Fake test doubles are a simplified version of the actual components, and they are fully functional but used only for testing [32].

Example:

In this example, we verify that the gateway returns False when making a payment with a negative amount 2.11 and utilise a fake 2.12 object to complete this test.

```
def test_make_payment_with_fake(self):
    gateway_fake = PaymentGatewayFake()
    processor = PaymentProcessor(gateway_fake)
    result = processor.make_payment(-100)
    assert result== False
```

Listings 2.11: Unit Test for Payment Processor with Fake Gateway

```
class FakePaymentGateway(PaymentGatewayInterface):
    def __init__(self) -> None:
        self.payments: List[int] = []

    def process_payment(self, amount: int) -> bool:
        if amount < 0:
            return False
        self.payments.append(amount)
        return True
```

Listings 2.12: Fake Payment Gateway Implementation

2.5.5 Snapshot testing

Snapshot testing is a test method that captures and compares snapshots by asserting that the generated image is the same as the reference image. This approach aims to ensure the consistency and correctness of the visual outputs and detect changes. It is commonly used in front-end development but can also be applied to other domains including document comparing [34].

Example

Scenario: Let's examine a Python function designed to resize an image:

```
def resizeImage(image_path, width, height):
    image = cv2.imread(image_path)
    resized_image = cv2.resize(image, (width, height))
    return resized_image
```

Listings 2.13: Code Under Test Example.

This snapshot test precisely verifies that the *resizeImage* function produces the intended output, ensuring the shape and pixel values of the resized image match those of the snapshot image 2.14.

```
def test_resize_image():
    input_image_path = "./input_image.jpg"
    snapshot_image_path = "./snapshot_image.jpg"

    resized_image = resizeImage(input_image_path, 300, 200)
    cv2.imwrite(saved_resized_image, resized_image)

    snapshot_image = cv2.imread(snapshot_image_path)
    resized_image_loaded = cv2.imread(saved_resized_image)

    assert resized_image_loaded.shape == desired_image.shape
    assert np.array_equal(resized_image_loaded == desired_image).
```

Listings 2.14: SnapShot Test.

Used Techniques in Image Snapshot Testing

There are several methods to measure the similarity between images in snapshot testing. Those are some of the most used techniques in this field:

- **Mean square error** : In image similarity, Mean Square Error (MSE) is a metric that detects the difference between two images. It measures the average squared difference of pixel intensities between corresponding pixels in two images [35].

  Lower MSE indicates a higher similarity between the two images being compared, as it suggests that the pixel values in the two images are closer to each other.

- **The Structural Similarity Index Measure (SSIM)** : The SSIM is a metric technique used to measure the similarity between two images. SSIM considers changes in structural information, luminance, and contrast that affect human visual perception [35].

  The SSIM index is based on three components:

  Luminance Comparison: Measures the similarity in luminance between corresponding pixels in the two images.

  Contrast Comparison: Compares the contrast of the two images. It captures the changes in contrast that are perceptible to humans.

  Structure Comparison: Evaluate the structural similarity between the two images. It considers the spatial patterns and arrangement of pixel intensities.

- **Histogram Correlation Similarity :** is a method used to compare the similarity between two images based on their histograms. However, in some cases, having similar histograms between images doesn't necessarily mean that those images are the same; the different images can have similar colour distributions sometimes [35].

2.6 Conclusion

In this chapter, we explained maintainable software and some techniques for developing it including SOLID principles. We presented a couple of metrics used to evaluate the code quality and introduced some usable design patterns. We also presented TDD to build robust and maintainable software. We also presented test doubles, used for testing components without using the implemented dependencies. We finished by presenting snapshot testing.

Chapter 3

Image Processing

3.1 Introduction

Most of the time, many image features are not noticeable due to the noise generated in the image-capturing and acquisition process. This minimizes the image values, especially if it would be used for training machine learning and deep learning models, here where the image processing appears.

Image processing is a field that is interested in manipulating digital images to extract meaningful information or enhance visual quality. Image processing helps us to analyze, modify, and interpret images in multiple contexts by employing algorithms and techniques. In section 3.2, we define the image and its fundamentals. In section 3.3, we present some of the most famous image processing operations.

3.2 Image Fundamentals

3.2.1 Definition of an Image

An image is a visual representation or painting of an object, scene, or person, typically captured, created, or stored in digital or analogic form for visual perception [36].

3.2.2 Definition of the digital Image

A digital image is a matrix of pixels, where each pixel contains fixed values about colour and intensity. Each image has a defined number of channels depending on its colour encoding [36].

the figures 3.1 and 3.2 shows how an image is represented.

- **Pixel:** The pixel is the smallest unit in a digital image that represents a single sample in the captured scene. It is typically a square or rectangular area in the image and has a specific value [37].
- **Channels:** A channel is an array of pixel values of an image representing one of the primary colours or other characteristics of an image. A grayscale is composed of one channel, whereas colour digital images are composed of multiple channels [35]. Some well-known colour models used to represent an image include Red-Green-Blue (RGB), Hue-Saturation-Value (HSV), Cyan-Magneta-Yellow-Black (CMYK), and others.

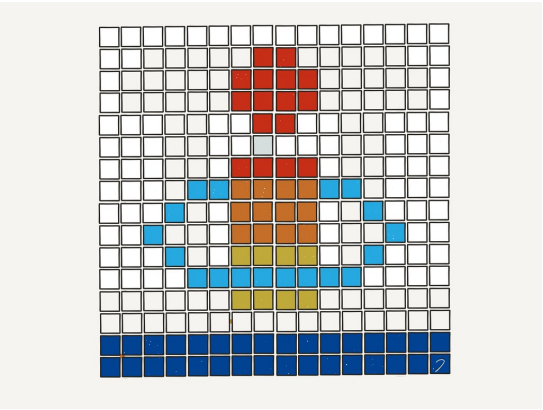

Figure 3.1: The Image [5].

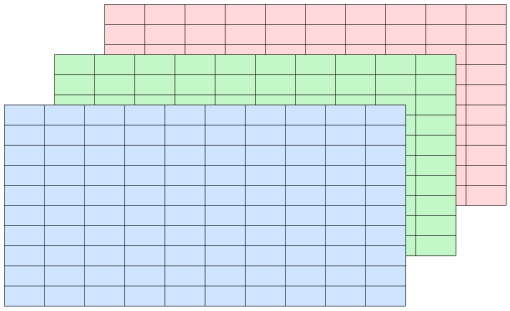

Figure 3.2: Pixels and Channels in Image.

3.2.3 Image Dimension:

The image dimension is the size of an image. It is represented by height and width, which essentially tells us the total number of pixels that usually impact an image's clarity and quality [36].

The dimension is usually expressed by the equation:

$$height \times width$$

For instance, an image with 400 pixels height and 300 pixels width would have dimensions of

$$400 \times 300$$

3.2.4 Resolution :

Image resolution is described by Pixels Per Inch (PPI). Higher resolutions mean more pixels per inch, while lower resolutions have few pixels, and the pixels become too large [36].

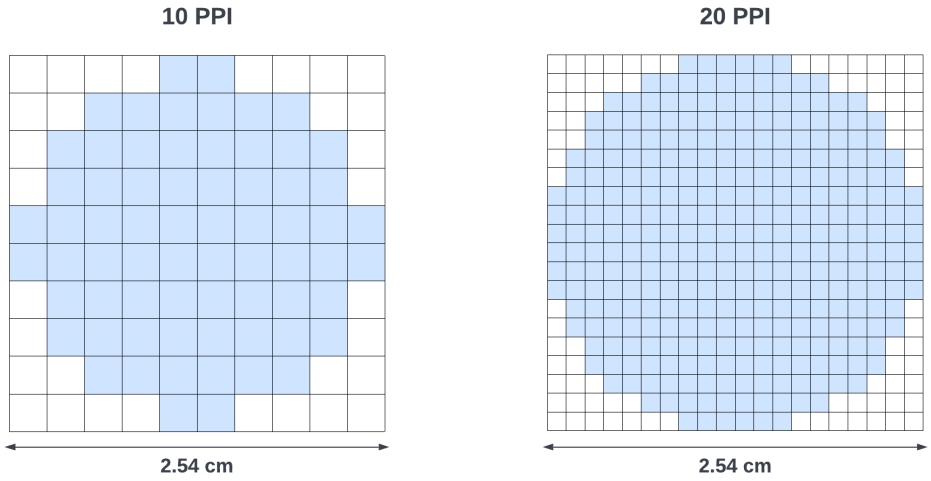

Figure 3.3: Image Resolution Comparison.

3.3 Image Processing

Image processing is a collection of operations to enhance the image quality or extract useful information. The input will be the image, and the output may be the image or a feature related to that image [35]. Various operations in image processing are used depending on the user's purpose and image status.

3.3.1 Image Resizing

Image resizing is changing the dimensions of an image, which includes scaling up or down the width and the height of an image by computing the pixel values for the newly resized image using some algorithms. There are multiple algorithms, but the core idea is interpolating the pixel [35].

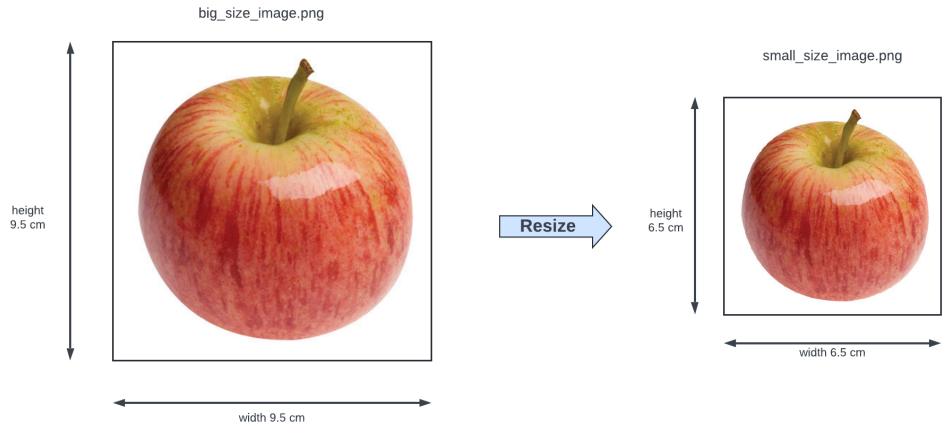

Figure 3.4: Image Resizing.

3.3.2 Image Segmentation

Image segmentation is dividing a digital image into segments or objects within the image. Pixels with similar characteristics are grouped, which helps identify and separate different elements within the image [35]. Figure 3.5 shows image segmentation applied to an image in the street; we see that it divides the various objects like cars, roads, and buses.

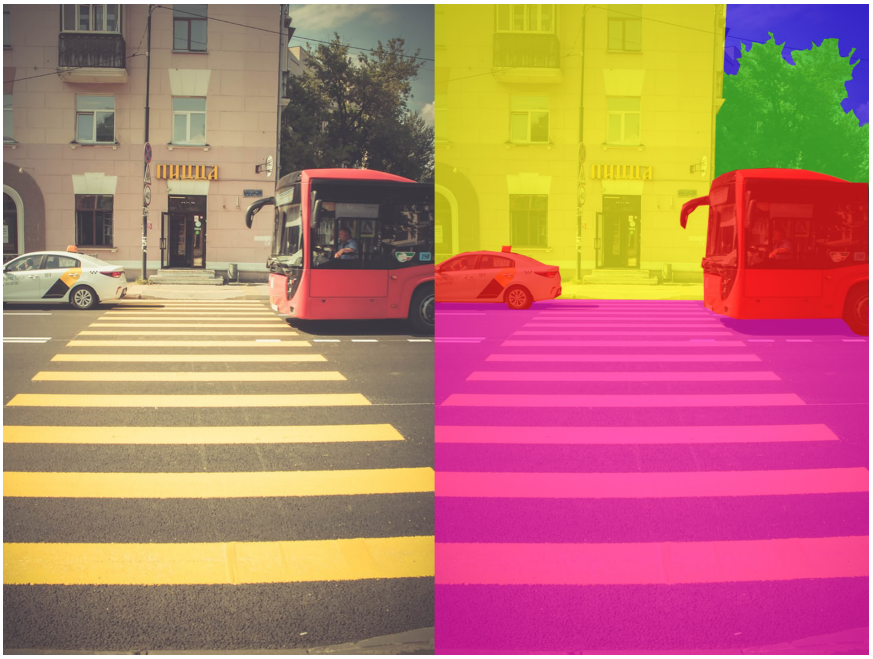

Figure 3.5: Image Segmentation [6].

3.3.3 RGB to Grayscale

RGB to Grayscale is a processing operation that converts an RGB colour image into a grayscale image, where pixel values range from black to white and shades of grey and are represented in a single channel [35].

Figure 3.6 shows RGB image converted to Grayscale.

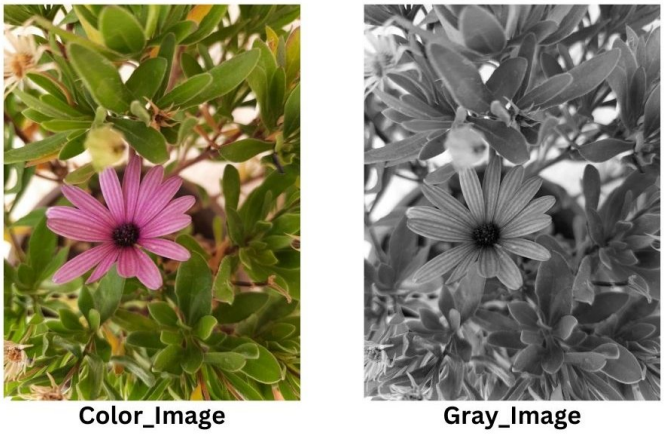

Figure 3.6: RGB to Grayscale Conversion.

3.3.4 Median Filter

Processing with a median filter is a technique that reduces noise in an image. It replaces each pixel's value with the median value of neighbouring pixels within a specified kernel size. The median filter effectively preserves edges and details in the image while reducing noise. It is commonly used in image processing applications where preserving image features is essential [35].

Figure 3.6 presents an example of noise removing from an image using a median filter.

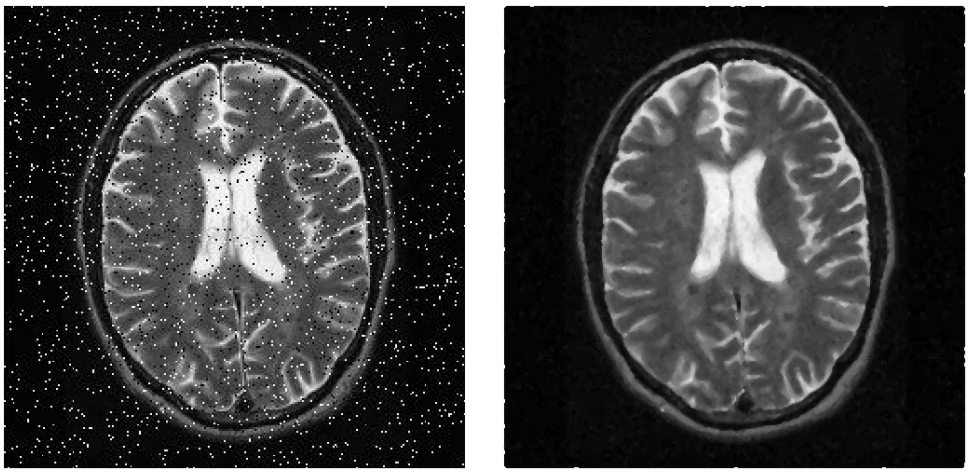

Figure 3.7: Noise Removal with Median Filter.

3.3.5 Histogram Equalisation

Histogram Equalisation is an operation to enhance the contrast in an image by stretching its histogram, where areas with lower contrast gain a higher contrast. It is widely used in medical image processing. It has side effects like increasing noise and over-enhance images, making them look unnatural [35].

Figure 3.8 shows an example of equalizing an image's histogram.

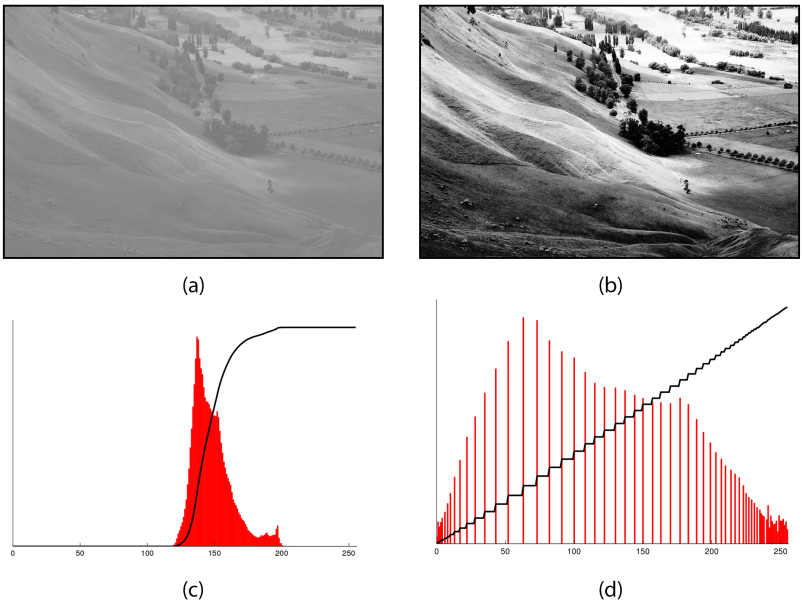

Figure 3.8: Histogram Equalisation [7].

3.3.6 Image Augmentation

Deep neural networks require a lot of training data to obtain good results and avoid overfitting in the model. Data augmentation techniques offer an excellent approach to prevent overfitting while effectively leveraging smaller datasets. Image augmentation is a process of creating new training data from the existing ones. To create new samples, you slightly change the original image. For instance, by making the new image a little brighter, mirroring the original, zooming it, etc [38].

3.4 Conclusion

In this chapter, we presented the image and its fundamentals, like pixels and channels, and showed some image processing operations and algorithms to enhance image quality. We saw that image processing may raise the values of images by showing hidden features and removing noise. It is essential for better understanding and explaining images and very helpful for building artificial intelligence models.

We aim to use what we learned in processing the dataset images to build a machine-learning model and find the best combination of operations that gives the best results in our model.

Chapter 4

Artificial Intelligence

4.1 Introduction

Artificial intelligence refers to the simulation of human intelligence processes by machines, especially computer systems. Within this domain are several subsets, each contributing to the overall landscape of intelligent systems. This chapter will explore two significant subsets: Machine Learning (ML) and Deep Learning (DL).

In section 4.2, we introduce machine learning and famous Machine Learning algorithms. In section 4.3, we discuss Deep Learning and present some famous Deep Learning algorithms. Then, in section 4.4, we present how to evaluate Machine Learning models. Lastly, in section 4.5, we present some challenges we may face in training an ML model.

4.2 Machine Learning

4.2.1 Definition

Machine learning is a branch of artificial intelligence where algorithms learn from data to make predictions or decisions without being explicitly programmed [38].
Machine learning is the art of programming computers to make them learn from data [38].

4.2.2 Machine Learning Tasks

Classification

Classification is the task of categorising input data into predefined classes or categories based on observed features. It is a type of supervised learning where the algorithm learns from labelled training data to make predictions on new unseen data [38].

Clustering

Clustering is an unsupervised learning technique that identifies similar instances and groups them into clusters [38].

Dimensionality Reduction

Dimensionality Reduction is a technique used to reduce the number of features and inputs of a dataset. It is widely used in training ML models with large datasets since it minimises computational costs and time by removing noise and redundant features, which improves model performance and accuracy. There are multiple algorithms used for this mission we cite: Principal Component Analysis (PCA), Locally Linear Embedding (LLE), and Multidimensional Scaling (MDS) [38].

Object Detection

Object detection is the task of classifying and defining the precise localisation of objects in an image or a video [38].

Image Capturing

Image captioning is the task of interpreting images with text. It works by extracting information with computer vision and describing it with text [38].

4.2.3 Common Machine Learning Algorithms

Support Vector Machines (SVM)

Support vector machine is a popular ML algorithm for classification. It identifies a hyperplane (a decision boundary) in a high-dimensional space that effectively distinguishes the data points belonging to various classes with the greatest margin [38].
Figure 4.1 is an illustration of how SVM works.

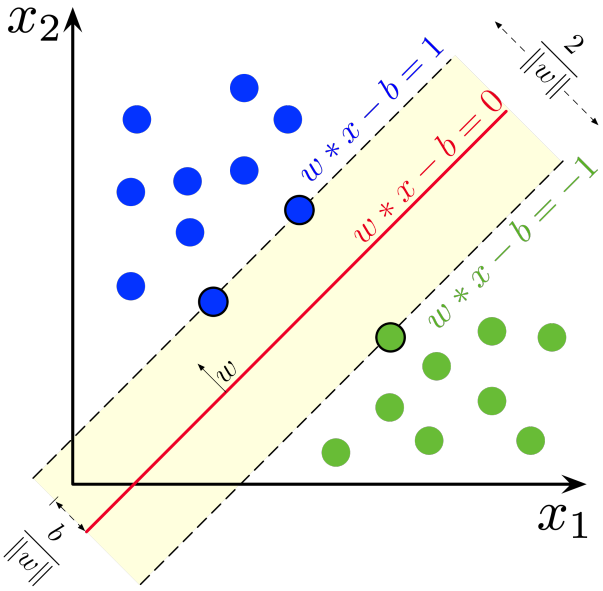

Figure 4.1: SVM [8].

K-Nearest Neighbours (KNN)

KNN is a supervised ML algorithm that classify new data points based on its neighbours points classes [38].

PCA

PCA is the most popular dimensionality reduction algorithm, it reduces the dimensionality and preserves the variability as much as possible. It is based on the projection approach. It finds a hyperplane near to the data and then projects the data in it [38].

K-Means

K-Means is an unsupervised ML algorithm for clustering. It groups similar data points into a predefined number of clusters. For each cluster, there is a centroid, which is the centre point of the cluster. The

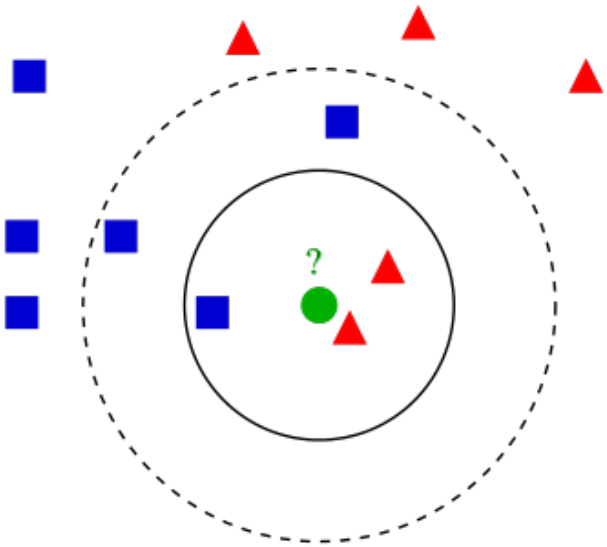

Figure 4.2: KNN [9].

centroids are placed randomly at the beginning, and they change their emplacement for each iteration by calculating how far the centroid is from the similar data points groups. This process repeats until it finds a good cluster with good results.

Figure 4.3 shows an example of three clusters adjusted through 9 iterations.

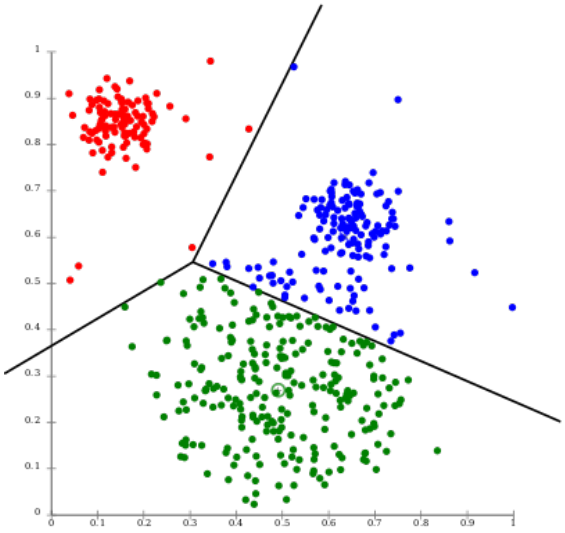

Figure 4.3: K-Means [10].

4.3 Deep Learning

4.3.1 Definition

Deep learning is a specific machine-learning technique It emulates the human brain's structure by containing multiple layers of interconnected processing units known as neurons. They use Artificial Neural

Networks (ANN) with deep architecture to solve complex problems and achieve complex pattern recognition. They are widely used in text, image, and audio detection or generation [39].

4.3.2 Algorithms of DL

Convolutional Neural Networks (CNN)

CNN is a deep learning model designed explicitly for processing structured grid data, like images or videos. It uses layers of filters to automatically learn features from the input data, enabling it to recognise patterns and make predictions. CNNs are widely used in computer vision tasks such as image recognition, object detection, and image classification [38].

Figure 4.4 illustrates a CNN sequence, highlighting the key stages involved in processing input data and generating output.

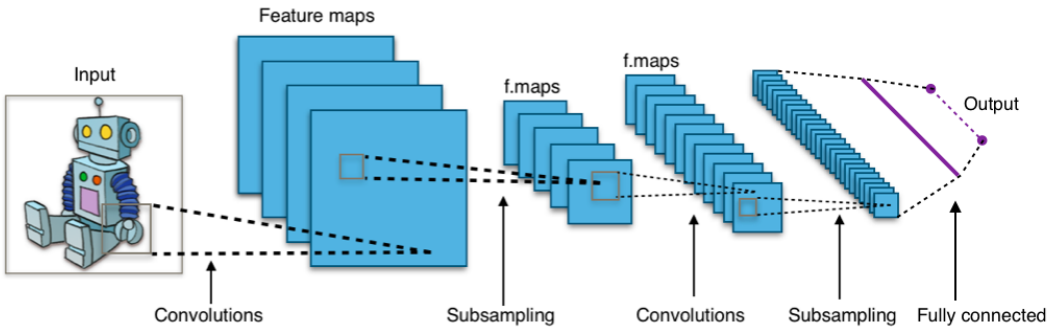

Figure 4.4: CNN [11].

Transfer Learning

Transfer learning is a technique used to build DL models by combining pre-trained models that are trained to classify any topic with other ANN. Transfer learning is famous for training with small datasets and achieving good performance. VGG16 and Inception v3 are common choices in transfer learning [38].

VGG16

VGG16 is a deep CNN architecture developed by the Visual Geometry Group (VGG) at the University of Oxford. It is characterised by its depth and simplicity, and it was one of the leading models in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2014 [38].

Inception v3

Inception v3 is a deep CNN architecture that is part of Google's Inception family of models. It is known for its high performance in image classification tasks and innovative design features [38].

You Only Look Once (YOLO)

YOLO is an open-source, real-time object detection algorithm. It is known for its speed and accuracy. It was introduced in 2015 and has numerous versions reaching YOLOV9 [38].

4.4 ML Models Evaluation Metrics

Model evaluation is a very important operation in making effective and accurate models. It helps us to measure the performance and validate models, and it allows us to check their situation and interpret them. There are multiple metrics used for this task, and for classification models we cite:

Accuracy

The accuracy is the percentage of the correct predictions made by the model over the testing data. It is generally not the preferred performance measure for classifiers, especially when using a dataset where classes do not have the same number of samples [38].

Confusion Matrix

The confusion matrix is a table that counts the number of samples predicted and their actual class. It is a better metric than accuracy since it gives much information about the correctness of the prediction and allows us to visualize it. However, it is not very easy and efficient to read [38].

Precision

Precision is the accuracy of positive predictions. It is calculated by the number of true positive predictions on all the positive predictions [38].

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

Recall

The recall is a metric that determines the percentage of positive samples that the model detects. It is also known as sensitivity, and it is calculated as the division of TP over the sum of TP and false negatives FN [38].

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

F1 Score

The F1 Score is a metric that combines precision and recall. It is calculated as the harmonic mean of precision and recall [38].

$$\text{F1 Score} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

4.5 Implementation Challenges

Overfitting

Overfitting is when a model fits close or exactly on the training dataset and performs well on training but performs badly on new data [38].

Underfitting

Underfitting is when a model does not learn on a dataset, leading to a bad performance in training and testing data. It may appear because the model is simple to learn or because of the poor quality of the dataset [38].

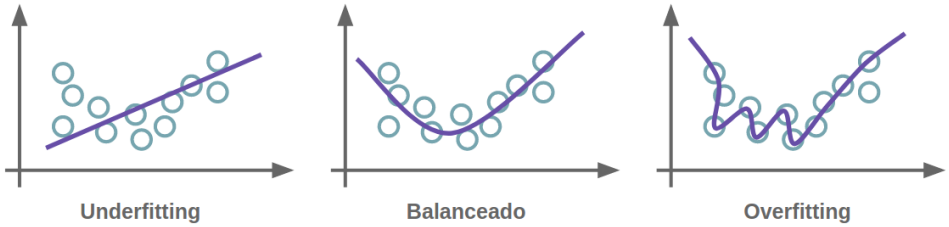

Figure 4.5: Overfitting and Underfitting [12].

Computational Resources

The limitation of computational resources is one of the biggest challenges ML researchers and engineers face, especially when training complex algorithms with large datasets, which would slow the training operation.

The input pipeline is an effective solution for this problem. It is a simple utility provided by multiple ML libraries like TensorFlow and Scikit-learn. It provides loading data in batches to avoid memory overloading. And provides parallel data loading and processing with model training which maximises the hardware resources utilisation.

4.6 Conclusion

In this chapter we introduced the machine learning and some famous techniques and some used algorithms for ML. later we presented deep learning and CNN as an algorithms of DL. Then we presented evaluation metrics to measure ML models performance. Lastly, we cited some famous challenges that we may face in building models.

Chapter 5

Contribution, Test, and Experiments

5.1 Introduction

In this chapter, we apply the research and topics that we discovered to build contributions. We begin in section 5.2 and present the OCT images dataset we built. In Section 5.3, we present the developed image processing framework by applying maintainable software techniques like TDD and clean code to process image datasets by creating operations modules we discussed in image processing. In section 5.4, we present the OCT ML models we built and compare their results and accuracy. Lastly, in section 5.5, we speak and show the app that helps users and doctors to use the model in an easy and friendly way.

5.2 Collected and used Datasets

We used two datasets to train and validate to build an ML model. The first one is collected from the Algerian-Cuban Friendship Hospital for Ophthalmology in ElOued. In contrast, the second one is available online [40] and named kermany2018.

5.2.1 Algerian-Cuban Friendship Hospital Dataset

Data Collection Process

Collecting the dataset was challenging, so we used a partner to help us: Algerian-Cuban Friendship Hospital For Ophthalmology In Eloued, see Figure 5.1. The dataset images were captured using an OCT machine and then labelled by a specialist from the hospital.

Figure 5.1: Algerian-Cuban Friendship Hospital For Ophthalmology In Eloued.

We ensured patients were fully aware of the data use and their rights to refuse when collecting the dataset. We took care of privacy and preserve personal data from unauthorised access. Additionally, we gathered only essential data to use in our operation.

The main originality of the collected dataset is the race of patients. This Arabian race retina would help build effective ML models on this race and can be used for multiple other uses since retinas change from one race to another.

Data Description and Exploratory Data Analysis (EDA)

Dataset Summary

The dataset summary provides an overview of the dataset, including the number of entries, types of data, and basic statistics. we have 256 different patients

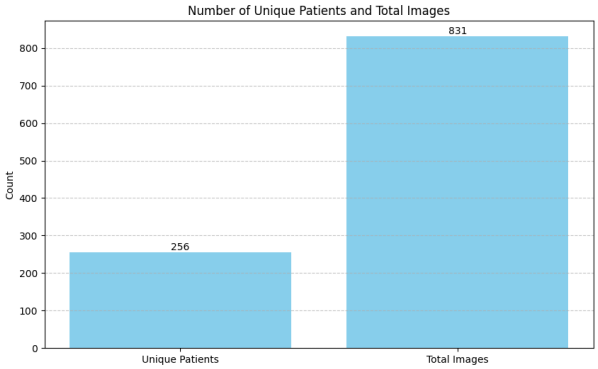

Figure 5.2: Summary of the dataset.

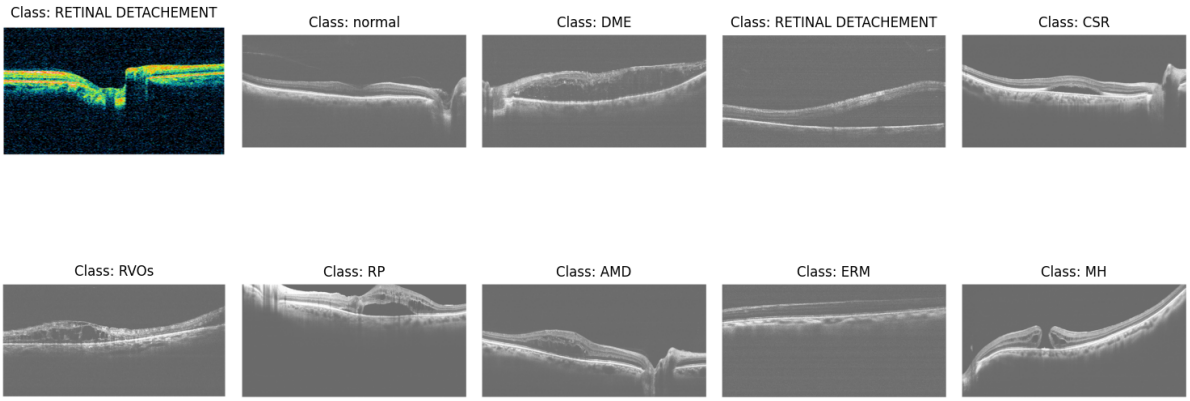

Figure 5.3: Summary of the dataset.

Classes Distribution

Graph 5.4 shows the dataset is not very balanced. We will try to balance it in the preprocessing phase to prevent the model from over-fitting.

Gender

This visualisation shows the gender distribution of patients, helping to understand the demographic breakdown of the dataset.

This visualisation shows the gender distribution of patients, helping to understand the demographic breakdown of the dataset.

Eye Positions

The distribution of eye positions helps understand the variety of data related to eye positions in the dataset.

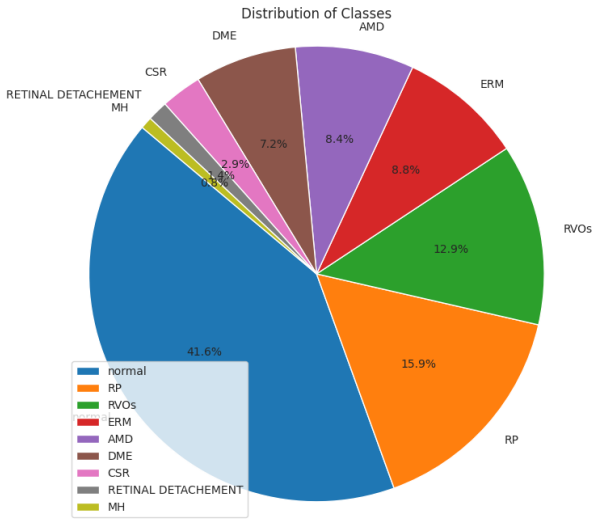

Figure 5.4: Distribution of classes in the dataset.

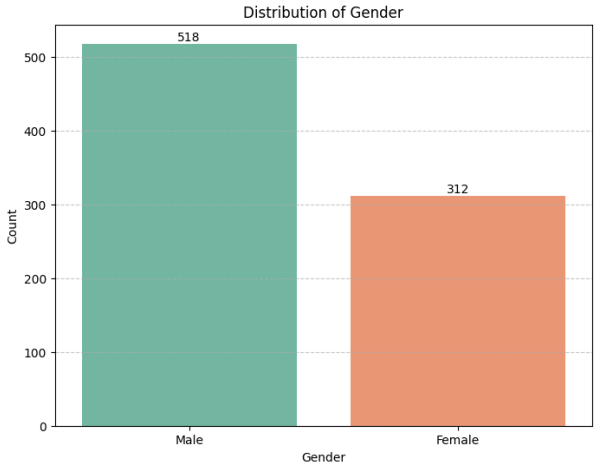

Figure 5.5: Gender distribution of patients.

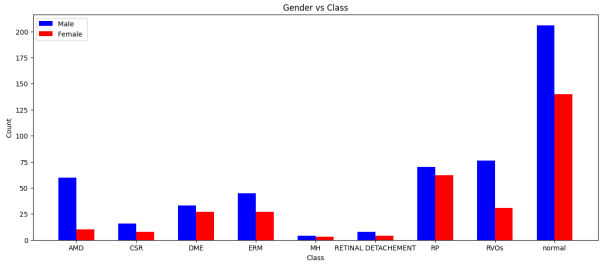

Figure 5.6: Gender distribution of patients.

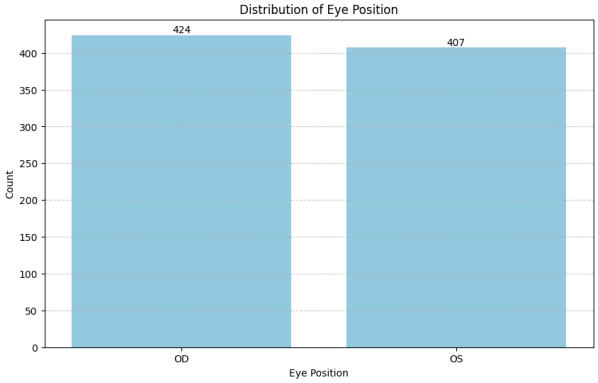

Figure 5.7: Distribution of eye positions.

Patients Ages

This section provides an overview of the age distribution of patients in the dataset.

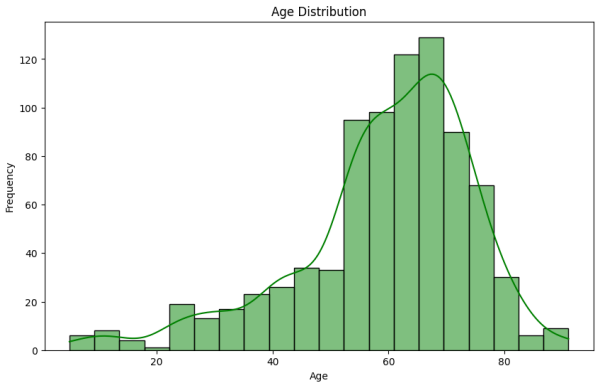

Figure 5.8: Age distribution of patients.

Distribution of Patient Age for Different Classes

This graph analyses how patient age varies across different classes, providing insights into age-related trends. The box plot compares the age distribution across different classes, helping to identify age-related patterns within each class.

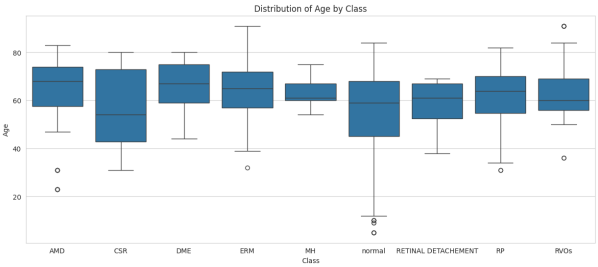

Figure 5.9: Distribution of patient age across different classes.

Clustering On Dataset

The collected dataset was misclassified at some situations, which gives wrong predictions and poor performance ML models. We employed clustering with K-means algorithm, and asked help from the experts at the hospital to re-categorise the data.

5.2.2 Kermany2018 Dataset

Kermany2018 dataset is available on Kaggle and includes more than 980 OCT images. These images do not detail the patients' genders, ages, or eye positions. It's divided into four classes, as illustrated in the figure 5.10.

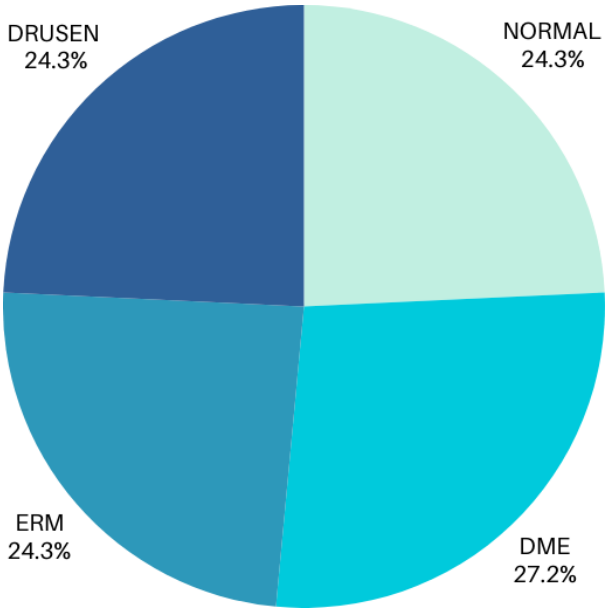

Figure 5.10: Distribution of classes in the kaggle dataset.

5.3 Developped Image Processing Framework

5.3.1 Introduction to Image Processing Framework

Image processing is important to improve image quality and extract useful information. This framework builds upon these principles, offering **CleanImageProcessing** as a solution to address specific challenges within image processing. It aims to build software that provides an all-in-one solution to process our images for training or validating ML and DL models. This framework focuses on maintainability and extensibility. Besides, it incorporates agile principles, such as Minimum Viable Product (MVP), continuous improvement and iterative development, to quickly adapt to changing requirements.

5.3.2 Architecture

The figure 5.11 shows the general architecture of the framework. The framework is modular, meaning we can change each component anytime. We have the ImageProcessorAppInterface component, which is the most stable since it does not depend on any other component. Then, we have the concrete component ImageProcessorApp, which respects the OCP from SOLID principles and depends on the abstraction of ImageProcessorInterface and ImagesGatewayInterface. LocalImagesGateway is a concrete implementation of ImagesGatewayInterface, and it is a component that respects SRP by having the responsibility of dealing with images only. ImageProcessor is a component that is the unit where the

image processing is done, and it injects the processing events Lastly, we have the Resizer component, which can be changed anytime without editing the existing code.

Figure 5.12 shows the Domain events pattern in action in our app. If we want to create a new processing operation, it only has to be a subclass of ProcessingEventInterface, so our app is open to extension respecting the OCP. This means new event types or handlers can be added without changing existing code.

In general, our code respects SOLID principles: the SRP by each component has a single responsibility, OCP all components are open to extend and close to modify, LSP is respected in the framework by for each subclasses do not lead to fatal errors in the app, it respects the ISP by depending on the abstraction of components, and lastly, the low-level components do not depend on high-level components which is a respect of DIP.

And that makes code cohesion and not coupled, and changing a component would not affect other components.

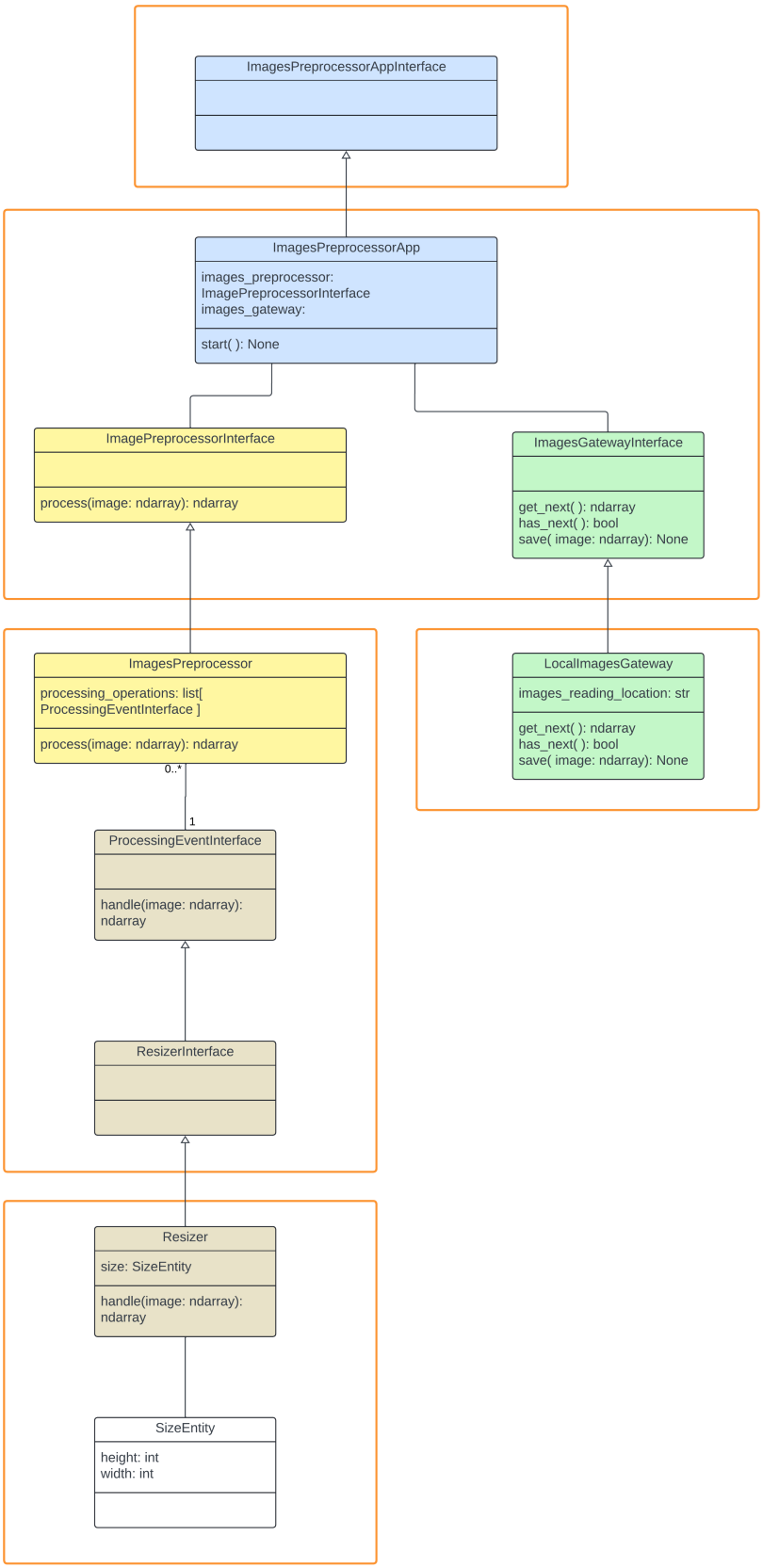

Figure 5.11: Image processing framework class diagram.

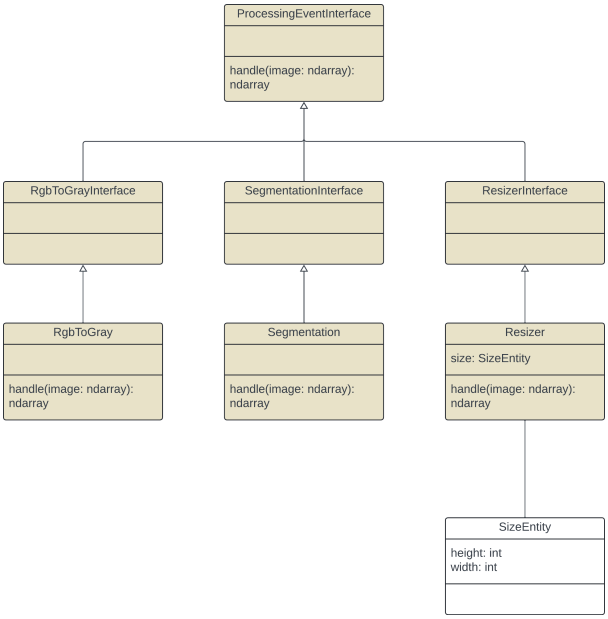

Figure 5.12: Domain events applied in the framework.

5.3.3 Processing Modules

Resizer

```python
from skimage.transform import resize
from app.entities.size_entity import SizeEntity
from app.modules.resizer.resizer_interface import ResizerInterface


class Resizer(ResizerInterface):
    def __init__(self, size: SizeEntity) -> None:
        def exception_on_different_type_of_size_parameter(__size: SizeEntity) ->
    None:
            if not isinstance(__size, SizeEntity):
                raise TypeError(
                "Size parameter have to be of the type SizeEntity and not None."
                )

        exception_on_different_type_of_size_parameter(size)
        self.size: SizeEntity = size

    def handle(self, image: ndarray) -> ndarray:
        return resize(image, (self.size.height, self.size.width))
```

Listings 5.1: Resizer Event Module.

Segmetation

```python
from numpy import ndarray
from skimage.util import img_as_ubyte
from skimage.filters import threshold_otsu
from app.modules.segmentation.segmentation_interface import SegmentationInterface


class SegmentationWithOtsuThreshold(SegmentationInterface):
    def handle(self, image: ndarray) -> ndarray:
        threshold = threshold_otsu(image)
```

```
        binary_image = image > threshold
        return img_as_ubyte(binary_image)
```

Listings 5.2: Segmentation Event Module.

RgbToGray

```
from numpy import ndarray
from skimage.util import img_as_ubyte
from skimage.color import rgb2gray
from app.modules.rgb_to_gray.rgb_to_gray_interface import RgbToGrayInterface


class RgbToGray(RgbToGrayInterface):
    def handle(self, image: ndarray) -> ndarray:
        return img_as_ubyte(rgb2gray(image))
```

Listings 5.3: RgbtoGray Event Module.

MedianFilter

```
from numpy import ndarray
from skimage.util import img_as_ubyte
from skimage.filters import median
from app.modules.median_filter.median_filter_interface import MedianFilterInterface


class MedianFilter(MedianFilterInterface):
    def handle(self, image: ndarray) -> ndarray:
        return img_as_ubyte(median(image))
```

Listings 5.4: MedianFilter Event Module.

HistogramEqualization

```
from numpy import ndarray
from skimage.util import img_as_ubyte
from skimage.color import rgb2gray
from skimage.exposure import equalize_hist
from app.modules.histogram_equalization.histogram_equalization_interface import (
    HistogramEqualizationInterface,
)


class HistogramEqualization(HistogramEqualizationInterface):
    def handle(self, image: ndarray) -> ndarray:
        gray_image = rgb2gray(image)
        equalized_image = equalize_hist(gray_image)
        return img_as_ubyte(equalized_image)
```

Listings 5.5: HistogramEqualization Event Module.

5.3.4 Compostition Root

```
from app.entities.size_entity import SizeEntity
from app.gateway.images_gateway_interface import ImagesGatewayInterface
from app.gateway.local_images_gateway import LocalImagesGateway
from app.image_preprocessors.image_preprocessor import ImagePreprocessor
from app.image_preprocessors.image_preprocessor_interface import (
```

```
    ImagePreprocessorInterface,
)
from app.images_preprocessor_app.images_preprocessor_app import
    ImagesPreprocessorApp
from app.modules.processing_event_interface import ProcessingEventInterface
from app.modules.resizer.resizer import Resizer
from app.modules.rgb_to_gray.rgb_to_gray import RgbToGray
from app.modules.median_filter.median_filter import MedianFilter
from app.modules.segmentation.segmentation_with_ostu_threshold import (
    SegmentationWithOtsuThreshold,
)


class ProcessingOperations:
    def __init__(self):
        resizer = Resizer(SizeEntity(width=10, height=10))
        rgb_to_gray = RgbToGray()
        median_filter = MedianFilter()
        segmentation = SegmentationWithOtsuThreshold()
        histogram_equalization = HistogramEqualization()

        operations_list: list[ProcessingEventInterface] = [
            resizer,
            rgb_to_gray,
            median_filter,
        ]
        image_preprocessor = ImagePreprocessor(
        processing_operations=operations_list
        )
        self.image_preprocessor: ImagePreprocessorInterface = image_preprocessor


class Gateway:
    def __init__(self):
        local_images_gateway = LocalImagesGateway(
            images_reading_location="",
            images_saving_location="",
        )
        self.gateway: ImagesGatewayInterface = local_images_gateway


class CleanImageProcessing:
    image_preprocessor = ProcessingOperations().image_preprocessor
    gateway = Gateway().gateway
    app = ImagesPreprocessorApp(
        images_gateway=gateway,
        image_preprocessor=image_preprocessor,
    )

    app.start()
```

Listings 5.6: Image Processing Framework Constructor Injection.

5.3.5 Code Analysis

Static Code Analysis

The framework analysis is shown in Table 5.1.

Metric	Production	Test	Total
Number of Classes	20	38	58
Number of Methods	39	449	488
Lines of Code	710	1685	2395
Source Lines of Code	404	1410	1814
Comments Lines	0	0	0

Table 5.1: Framework code static analysis metrics.

Test Coverage

The framework code is 100 % test-covered, which increases confidence in our code and helps in refactoring and later change. Figure 5.13 shows the coverage report generated by covrage.py library.

Coverage report: 100%

coverage.py v7.4.4, created at 2024-06-11 16:40 +0100

Module	statements	missing ↓	excluded	coverage
app/__init__.py	0	0	0	100%
app/entities/__init__.py	0	0	0	100%
app/entities/size_entity.py	24	0	0	100%
app/gateway/__init__.py	0	0	0	100%
app/gateway/images_gateway_interface.py	11	0	0	100%
app/gateway/local_images_gateway.py	63	0	0	100%
app/gateway/reading_gateway/__init__.py	0	0	0	100%
app/gateway/reading_gateway/reading_gateway_interface.py	7	0	0	100%
app/gateway/saving_gateway/__init__.py	0	0	0	100%
app/gateway/saving_gateway/saving_gateway_interface.py	5	0	0	100%
app/image_preprocessors/__init__.py	0	0	0	100%
app/image_preprocessors/image_preprocessor.py	28	0	0	100%
app/image_preprocessors/image_preprocessor_interface.py	5	0	0	100%
app/images_preprocessor_app/__init__.py	0	0	0	100%
app/images_preprocessor_app/images_preprocessor_app.py	12	0	0	100%
app/images_preprocessor_app/images_preprocessor_app_interface.py	3	0	0	100%
app/modules/__init__.py	0	0	0	100%
app/modules/histogram_equalization/__init__.py	0	0	0	100%
app/modules/histogram_equalization/histogram_equalization.py	10	0	0	100%
app/modules/histogram_equalization/histogram_equalization_interface.py	3	0	0	100%
app/modules/median_filter/__init__.py	0	0	0	100%
app/modules/median_filter/median_filter.py	7	0	0	100%
app/modules/median_filter/median_filter_interface.py	3	0	0	100%
app/modules/processing_event_interface.py	5	0	0	100%
app/modules/resizer/__init__.py	0	0	0	100%
app/modules/resizer/resizer.py	14	0	0	100%
app/modules/resizer/resizer_interface.py	3	0	0	100%
app/modules/rgb_to_gray/RgbToGray.py	7	0	0	100%
app/modules/rgb_to_gray/__init__.py	0	0	0	100%
app/modules/rgb_to_gray/rgb_to_gray_interface.py	3	0	0	100%
app/modules/segmentation/__init__.py	0	0	0	100%
app/modules/segmentation/segmentation_interface.py	3	0	0	100%
app/modules/segmentation/segmentation_with_otsu_threshold.py	11	0	0	100%
Total	**227**	**0**	**0**	**100%**

coverage.py v7.4.4, created at 2024-06-11 16:40 +0100

Figure 5.13: Framework test coverage.

Code Style

The code is readable and follows the PEP 8 Python style guide we showed before, which facilitates any developer to change and improve it.

5.4 The Proposed OCT Interpretation Models

5.4.1 CNN

This section provides a detailed overview of CNN architecture and the training process used in our experiments. The architecture of the CNN model and the steps involved in the training process are summarised in Tables 5.2 and 5.3, respectively. These tables describe each component and the corresponding hyper-parameters, loss functions, and regularisation techniques applied to the CNN model.

CNN Model Architecture

Component	Description
Input Layer	Expects images with dimensions `(128, 128, 1)`
Conv Layer 1	32 filters, 3x3 kernel, ReLU activation, same padding
Max Pooling Layer 1	2x2 pool size.
Conv Layer 2	64 filters, 3x3 kernel, ReLU activation, same padding
Max Pooling Layer 2	2x2 pool size
Conv Layer 3	128 filters, 3x3 kernel, ReLU activation, same padding
Max Pooling Layer 3	2x2 pool size.
Flatten Layer	Converts 2D feature maps into a 1D feature vector
Dense Layer 1	128 units, ReLU activation
Output Layer	4 units (assuming a 4-class classification problem), softmax activation

Table 5.2: CNN Model Architecture.

Training Process 5.3

Component	Description
Data Preprocessing	**Normalization**: Scale pixel values to [0, 1]. **Resizing:** resize images to 150x150 pixel
Hyperparameters	**Batch Size**: 30 **Number of Epochs**: 25 **Learning Rate**: 0.0001 (for Adam optimizer).
Loss Function	**Categorical Cross-Entropy**: Measures the difference between true and predicted label distributions.
Regularization Techniques	**Dropout**: Can be added to fully connected layers to prevent overfitting **Weight Decay**: Can be applied to the optimizer to constrain the magnitude of the weights.
Model Compilation	**Optimizer**: Adam **Loss Function**: Categorical Cross-Entropy **Metrics**: Accuracy
Model Training	**Fit the Model**: Train on training data with validation split to monitor performance. **Validation Split**: 20%

Table 5.3: CNN Model Training Process.

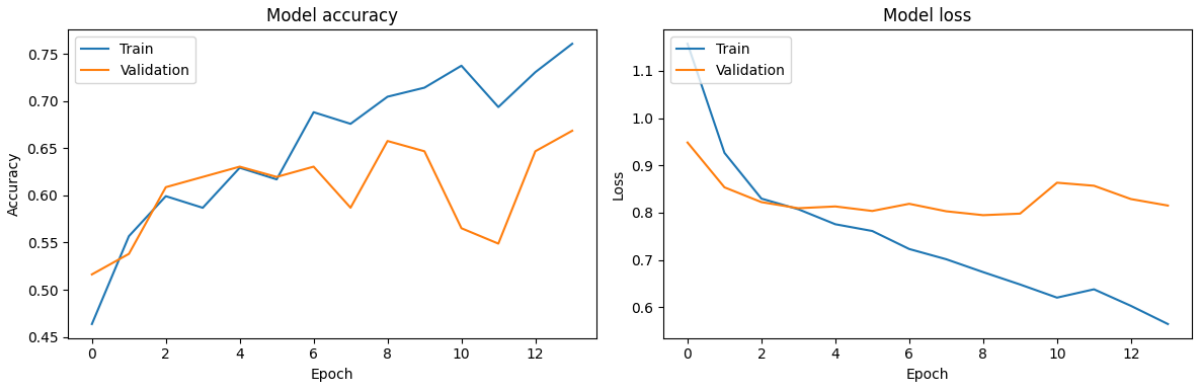

Figure 5.14: CNN Model loss and Accuracy over Epochs

Accuracy and Loss Plots

Confusion Matrix

The confusion matrix provides a detailed breakdown of the model's performance across different classes, as shown in figure 5.15.

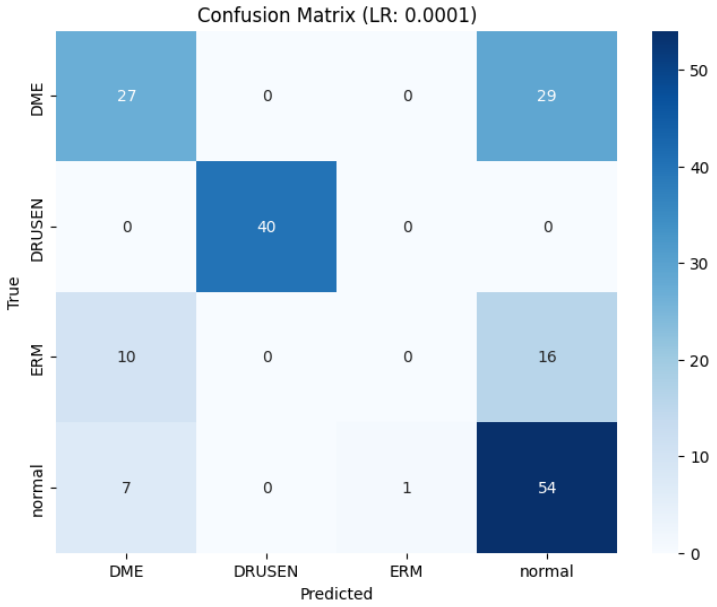

Figure 5.15: Confusion Matrix for Test Data

Result and Discussion

We have experimented with numerous CNN models, applying various parameter adjustments. The results of these experiments are detailed in Table 5.4. Table 5.5 highlights the details of the model with the highest accuracy.

Models	Layers	Filters per Layer	Kernel Size	Learning Rate	Dropout Rate	Accuracy
1	2	[32, 64]	5x5	0.0001	0.25	0.65%
2	3	[32, 64]	3x3	0.0001	0.25	0.65%
3	3	[32, 64, 128]	3x3	0.0001	0.25	0.66%
4	3	[32, 64, 128]	5x5	0.0001	0.5	0.57%
5	4	[32, 64, 128, 256]	5x5	0.0001	0.5	0.57%

Table 5.4: Model Parameter Changes and Corresponding Accuracy

Class	**Precision**	**Recall**	**F1-Score**	**Support**
DME	0.61	0.48	0.54	56
DRUSEN	1.00	1.00	1.00	40
ERM	0.00	0.00	0.00	26
Normal	0.55	0.87	0.67	62
Train Accuracy	0.66			
Validation Accuracy	0.66			

Table 5.5: Best Model Result.

5.4.2 Transfer learning

we used two pre-trained convolutional neural networks, VGG16 and Inception V3, to improve our model's performance. We aimed to enhance our results using these pre-trained models while saving training time and computational resources.

Inception V3:

The performance of the Inception V3 pre-trained model on the OCT dataset is illustrated through the confusion matrix and the accuracy and loss plots presented in Figures 5.17, 5.16.

Visual Geometry Group 16 (VGG16):

The results of the VGG16 pre-trained model are clearly illustrated in the confusion matrix (Figure 5.19) and the accuracy and loss plots (Figure 5.18).

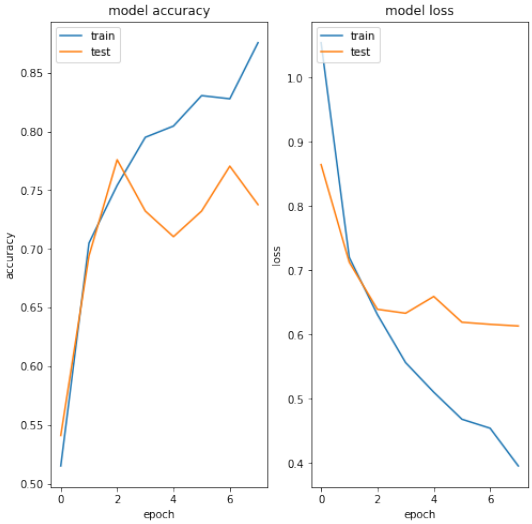

Figure 5.16: Inception V3 accuracy and loss

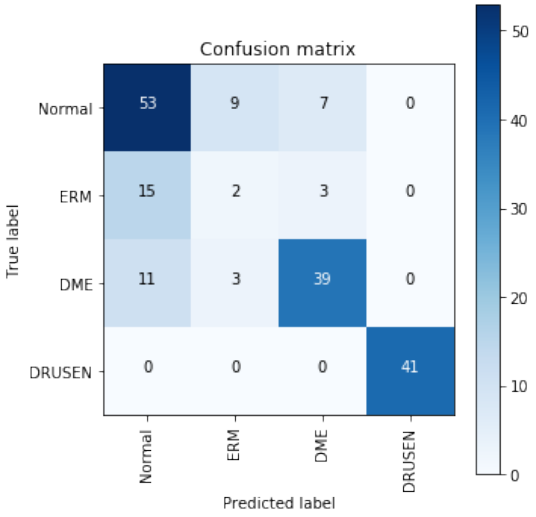

Figure 5.17: Inception V3 Confusion Matrix

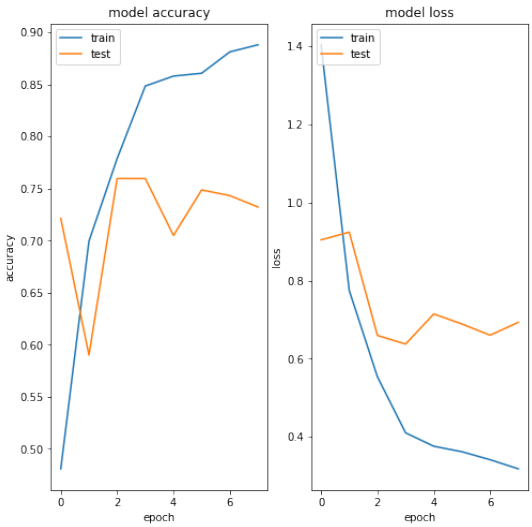

Figure 5.18: VGG16 accuracy and loss

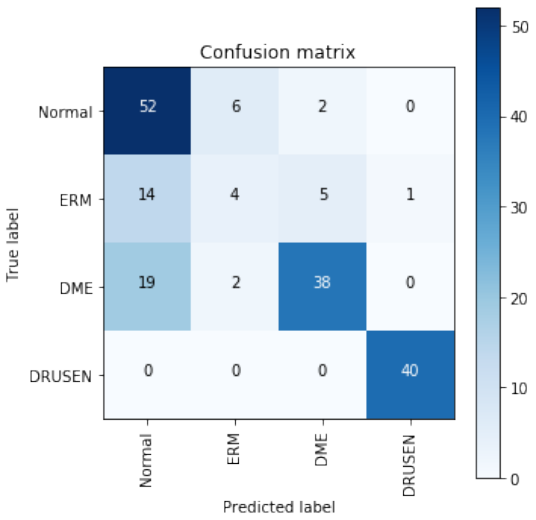

Figure 5.19: VGG16 Confusion Matrix

Result and Discussion

Class	Precision	Recall	F1-Score	Support
Normal	0.67	0.77	0.72	69
ERM	0.14	0.10	0.12	20
DME	0.80	0.74	0.76	53
DRUSEN	1.00	1.00	1.00	41
Train Accuracy	0.8757			
Validation Accuracy	0.7377			

Table 5.6: Inception V3 Model Results Details

Class	Precision	Recall	F1-Score	Support
Normal	0.61	0.87	0.72	60
ERM	0.33	0.17	0.22	24
DME	0.84	0.64	0.73	59
DRUSEN	0.98	1.00	0.99	40
Train Accuracy	0.8883			
validation Accuracy	0.7322			

Table 5.7: VGG16 Model Results Details

The Inception V3 and VGG16 models have close performance accuracies of 73.77% and 73.22%. Both models excel in predicting 'Normal' and 'DRUSEN' classes but struggle with 'ERM', highlighting the need for more data or fine-tuning for this class. Inception V3 shows a balanced performance across all classes.

5.4.3 YOLOv8

YOLOv8 is a very powerful tool in object detection. We use it to train a model that helps detect the boundary of the disease in the OCT images. The model results are as follows:

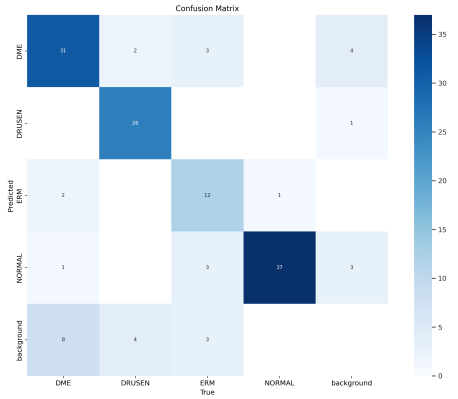

Figure 5.20: 4 Classes Confusion Matrix

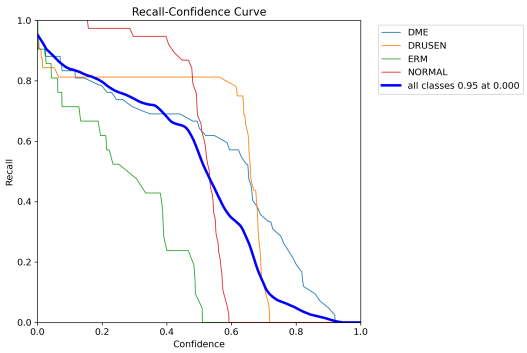

Figure 5.21: Recall

The YOLO8 model is doing well overall, better than others (CNN, VGG16, Inception V3, KNN, SVM with PCA) models. It gives high recall and precision and performs well with the ERM disease.

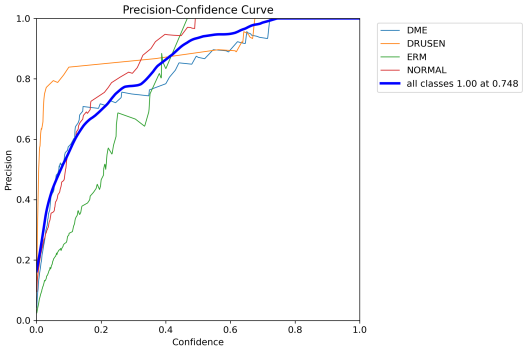

Figure 5.22: Precision

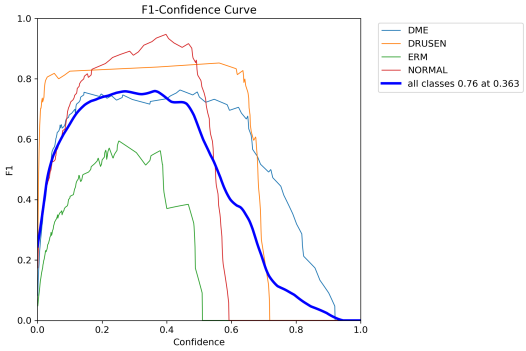

Figure 5.23: F1-Score

5.4.4 SVM with PCA

We trained an SVM model with PCA for feature extraction, used multiple component numbers, and got the accuracy results shown in figure 5.8.

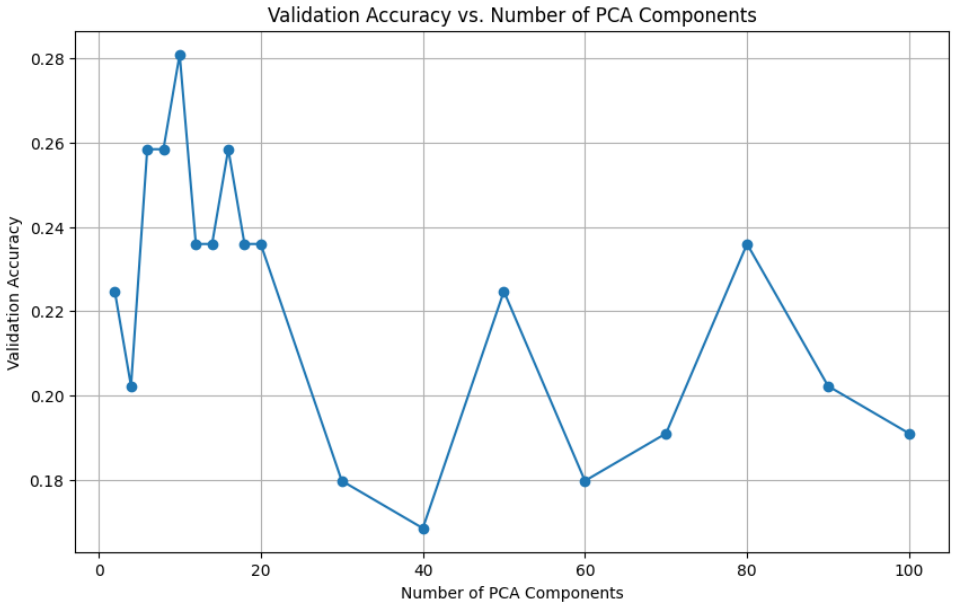

Figure 5.24: SVM with PCA.

The optimal solution was 0.2809 with ten components on PCA 5.8. This result is very low, so we try to use another algorithm. The table 5.9 shows detailed values of different metrics for each class.

Metric	Value
Number of PCA Components	10
Validation Accuracy	0.2809

Table 5.8: SVM with PCA best result.

5.4.5 KNN

We minimised the number of classes and trained a KNN model with the CSR, MH, RETINAL DETACHMENT, AMD, RP and ERM classes. In this experience, we didn't use PCA. We used multiple neighbouring experiences and got the results in figure 5.25.

Classification Report				
Class	**Precision**	**Recall**	**F1-Score**	**Support**
AMD	0.20	0.06	0.10	16
DME	0.43	0.20	0.27	15
ERM	0.20	0.40	0.27	10
RP	0.12	0.11	0.12	9
RVOs	0.30	0.53	0.38	15
Normal	0.36	0.33	0.35	24

Table 5.9: Classification Report for the Best Validation Accuracy

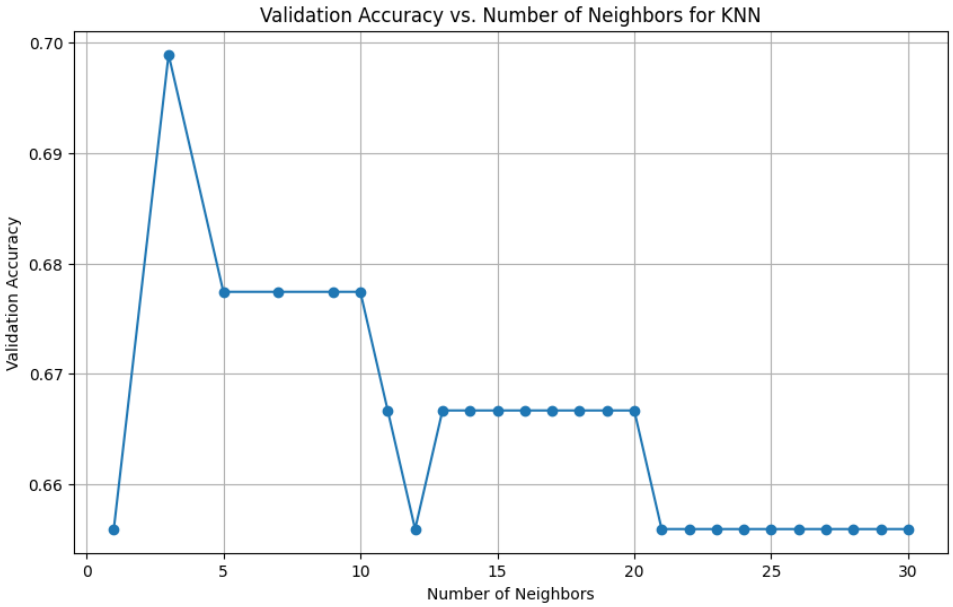

Figure 5.25: KNN with multiple neighbouring experiences.

We achieved a maximum accuracy of 0.6989 with three neighbours, which is a good result.

5.4.6 KNN with PCA

In this experience, we built a KNN model with different PCA components and neighbour numbers. We got the accuracy's shown in Figure 5.26.

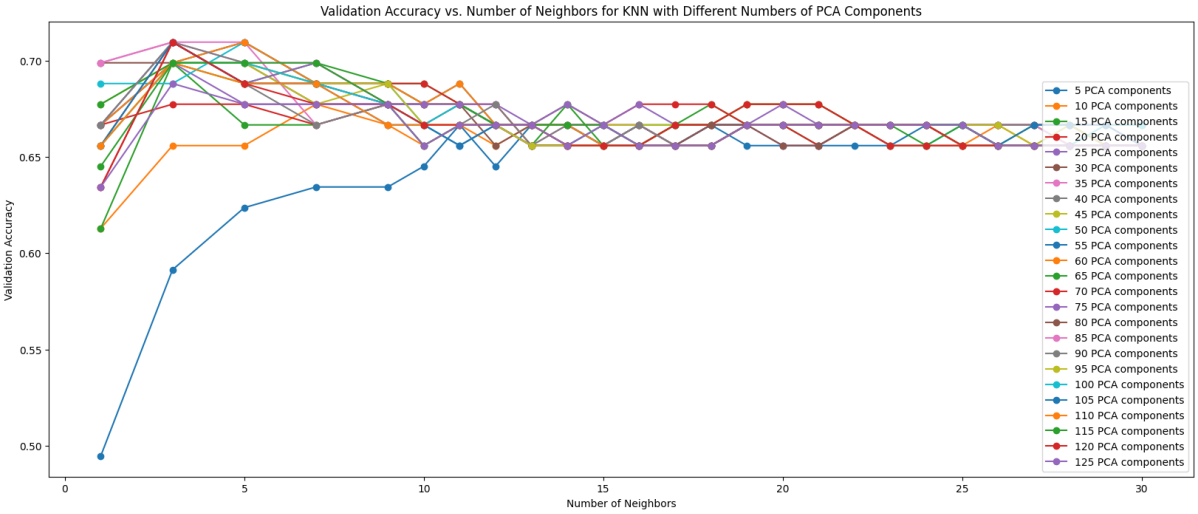

Figure 5.26: KNN and PCA with multiple neighbouring and components number.

The optimal result was 0.7097, achieved with 3 neighbours and 35 PCA components.

5.4.7 Final Results Analysis and Comparison

Model	Accuracy
CNN	0.66
YOLO	0.746
SVM with PCA	0.2809
KNN	0.6989
KNN with PCA	0.7097
VGG16	0.7322
Inception	0.7377

Table 5.10: Comparison of Model Accuracies

In analyzing results, YOLO performs the best with an accuracy of 0.746, followed by Inception and VGG16 at 0.7377 and 0.7322, respectively. CNN shows a decent performance with an accuracy of 0.66. KNN with PCA outperforms standard KNN, achieving 0.7097 versus 0.6989. The SVM with PCA has the lowest accuracy at 0.2809.

5.5 The developed Deployment Framework

5.5.1 Introduction

In this section, we build a Graphical framework to integrate with our OCT image detector model, which is very important for the doctors and the medical staff and makes them use the model quickly.

5.5.2 Design and Process

We focused on making the interface easy to use and effective. We followed a flexible development approach and created a basic application version as a Minimum Viable Product (MVP). We ensure the application works well on different devices and screen sizes, so we chose a service model called Streamlit that do all of this.

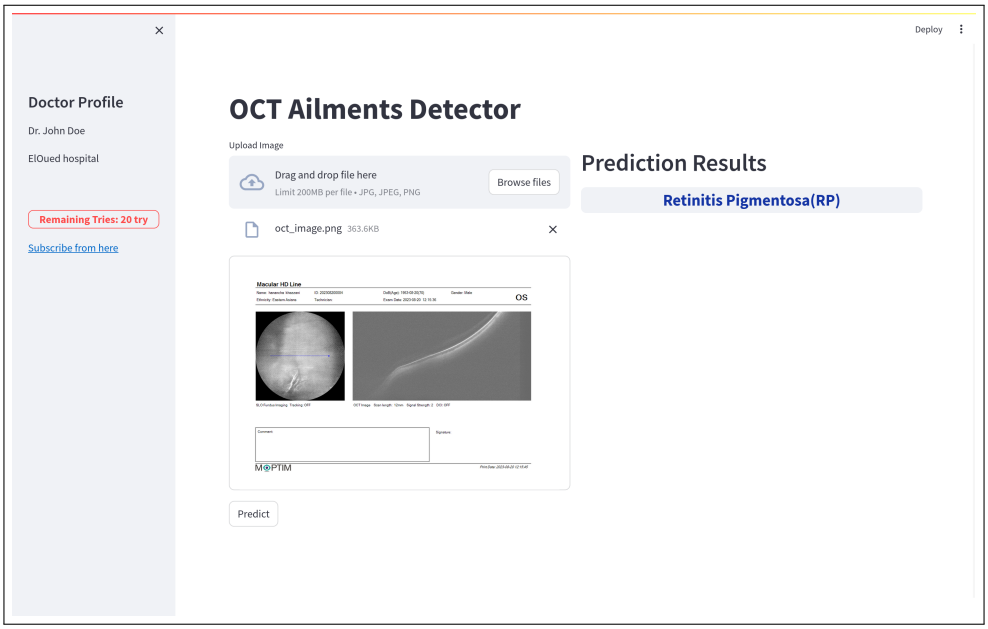

Figure 5.27: Framework UI.

The user uploads an OCT image using the image uploader component and then presses the predict button. The image is processed and predicted using the model. Finally, the framework presents the result.

5.5.3 Integration with Image Detection Model

We respected the SOLID principles when deploying a prediction model. There is no need to build a new UI; we inject the model, and the new model is used by our framwork.

```
class Model:
    def __init__(self, model_path, classes_names: list[str]) -> None:
        def import_model(model_path):
            """ importing the model and setting it in model attribute """

        self.model_path = model_path
        self.classes_names = classes_names
        self.model = import_model(model_path = model_path)

    def predict(self, image) -> str:
        model_prediction_result = self.model.predict(image)
        return self.classes_names[model_prediction_result]

def app_costructor_injection():
    model = Model("model",
                  ['Dry Eye Syndrome', 'Glaucoma', 'Cataracts',
                   'Macular Degeneration', 'Retinal Detachment',
                   'Diabetic Retinopathy', 'Conjunctivitis (Pink Eye)',
                  ])
    app = app(title = 'OCT Ailments Detector', model = model)
```

```
21      app.start()
```

Listings 5.7: Injection of the Model Into the App.

5.6 Conclusion

At the end of this chapter, we built a dataset of OCT images that will be used to build an ML model for detecting ocular ailments. We presented how we gathered the dataset, exploded it, and checked its variance. After that, we presented preprocessing operations on the dataset. Then, we built an Image Processing framework focusing on maintainability and extensibility and aiming to process digital images effectively. We presented the general architecture of the framework. We also presented some available processing modules. After, we presented the composition root and constructor injection and showed how to use the framework. Then, we analysed and evaluated our code. Lastly, we built a graphical framework to integrate the model, making users use it easily. In the future, we aim to add more functionality and make the app more practical and commercial.

Conclusion and Future works

In conclusion, even with the advancements in medical imaging technology, accurate and efficient diagnosis of ocular conditions remains a significant challenge. OCT, just like any ophthalmology imaging technique, provides high-resolution images of the eye's internal structures, but it still faces limitations in interpretation and analysis. The complexity of ocular anatomy and the variability in disease manifestations demand sophisticated image-processing techniques and reliable diagnostic algorithms.

Moreover, the development of maintainable software is crucial for ensuring the long-term effectiveness and adaptability of tools. Current software often lacks robustness and scalability.

Contributions

The research has been structured around four primary objectives: firstly, collecting a comprehensive dataset of OCT images specific to ocular conditions on individuals of Arabian descent; secondly, developing an advanced image processing framework that attaches to sustainable software practices; thirdly, training a machine learning model capable of predicting various ocular ailments from OCT scans; and finally, creating a flexible and user-friendly deployment framework to facilitate the integration and utilisation of the developed model by healthcare professionals and medical staff. Each thread contributes significantly to advancing the field of AI-driven medical diagnostics, aiming to improve the precision, efficiency, and accessibility of early disease detection and treatment planning.

Perspective and Future Works

Although we consider that the goal of the thesis has been accomplished, we could make many improvements to achieve better results. We aim in the future to accomplish these cited goals:

- Expanding the dataset with additional images.
- Utilising the dataset to construct other machine learning models for detecting different diseases.
- Fine-tuning the machine learning OCT model.
- Deploying the image processing framework online for open-source usage.
- Adding a simple graphical user interface to the image processing framework.
- Providing multiple image processing modules.

Bibliography

[1] Wikimedia Commons contributors. Lateral orbit anatomy 2, 2016. Accessed: 2024-06-06.

[2] Nursing Hero. Special senses - vision, n.d. Accessed: 2024-06-06.

[3] Wikimedia Commons contributors. Macula histology oct, 2024. Accessed: 2024-06-30.

[4] M. Seemann and S. van Deursen. *Dependency Injection Principles, Practices, and Patterns*. Manning, 2019.

[5] $\text{Metsuke}_e s. Untitled, 2016. Accessed : 2024 - 06 - 30.$

[6] Wikimedia Commons contributors. Image segmentation, 2007. Accessed: 2024-06-09.

[7] Wikimedia Commons contributors. Histogram equalization, 2007. Accessed: 2024-06-09.

[8] Wikimedia Commons contributors. Svm margin, 2024. Accessed: 2024-06-30.

[9] Wikimedia Commons contributors. Knn class, 2024. Accessed: 2024-06-30.

[10] Wikimedia Commons contributors. Kmeans gaussian data, 2024. Accessed: 2024-06-30.

[11] Wikimedia Commons contributors. Typical cnn, 2024. Accessed: 2024-06-30.

[12] Wikimedia Commons contributors. Underfitting and overfitting, 2024. Accessed: 2024-06-30.

[13] Qi Li, Shuo Li, Zhi He, Huanyu Guan, Rui Chen, Yihong Xu, and Wenying Wang. Deepretina: layer segmentation of retina in oct images using deep learning. *Translational vision science & technology*, 9(2):61–61, 2020.

[14] Rafal K. Ara, Adam Matiolanski, Andrzej Dziech, Radoslaw Baran, Piotr Domin, and Andrzej Wieczorkiewicz. Fast and efficient method for optical coherence tomography images classification using deep learning approach. *Sensors*, 22(13):4675, 2022.

[15] T.F. Freddo and E. Chaum. *Anatomy of the Eye and Orbit: The Clinical Essentials*. Wolters Kluwer, 2018.

[16] N. Yoshimura and M. Hangai. *OCT Atlas*. Springer Berlin Heidelberg, 2014.

[17] Daniel Epshtein. *What Does A Normal OCT Look Like?* https://eyesoneyecare.com/resources/what-does-a-normal-oct-look-like/.

[18] eye guru org. *How to read OCTs: 8 fundamental diseases*. https://eyeguru.org/essentials/interpreting-octs/?fbclid=IwAR1F7wLe6uDIcpirYX0odvOVOJkZ01b38Hd7f0V9GDD-aHSUNt0dGtDLeig.

[19] Muhammad Awais, Henning Müller, Tong Boon Tang, and Fabrice Meriaudeau. Classification of sd-oct images using a deep learning approach. In *2017 IEEE International Conference on Signal and Image Processing Applications (ICSIPA)*, pages 489–492. IEEE, September 2017.

[20] Christopher S. Lee, David M. Baughman, and Andrew Y. Lee. Deep learning is effective for classifying normal versus age-related macular degeneration oct images. *Ophthalmology Retina*, 1(4):322–327, 2017.

[21] Thomas Schlegl, Sebastian M. Waldstein, Hrvoje Bogunovic, Felix Endstraßer, Amir Sadeghipour, Anna M. Philip, and Ursula Schmidt-Erfurth. Fully automated detection and quantification of macular fluid in oct using deep learning. *Ophthalmology*, 125(4):549–558, 2018.

[22] Di Wang and Liansheng Wang. On oct image classification via deep learning. *IEEE Photonics Journal*, 11(5):1–14, 2019.

[23] Marek Pekala, Neil Joshi, Tingying Ashley Liu, Neil M. Bressler, Delia C. DeBuc, and Philippe Burlina. Deep learning based retinal oct segmentation. *Computers in biology and medicine*, 114:103445, 2019.

[24] Roger S. Pressman. *Software Engineering: A Practitioner's Approach*. McGraw-Hill Education, 9 edition, 2019.

[25] R.C. Martin. *Clean Code: A Handbook of Agile Software Craftsmanship*. Robert C. Martin series. Prentice Hall, 2009.

[26] R.C. Martin. *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Martin, Robert C. Prentice Hall, 2018.

[27] GeeksforGeeks. *Single Responsibility in SOLID Design Principle*. https://www.geeksforgeeks.org/single-responsibility-in-solid-design-principle/.

[28] Samuel Oloruntoba and Anish Singh Walia. *SOLID: The First Five Principles of Object-Oriented Design*. https://www.digitalocean.com/community/conceptual-articles/s-o-l-i-d-the-first-five-principles-of-object-oriented-design.

[29] Sam Millington. *A Solid Guide to SOLID Principles*. https://www.baeldung.com/solid-principles.

[30] Aleksandra Shershen. *SOLID principles: implementation and examples in C++*. https://medium.com/@aleksandra_shershen/solid-principles-implementation-and-examples-in-c-99f0d7e3e868.

[31] Lucas Porfirio. *SOLID Principles: They're Rock-Solid for Good Reason!* https://dev.to/lukeskw/solid-principles-theyre-rock-solid-for-good-reason-31hn.

[32] K. Beck. *Test-driven Development: By Example*. Addison-Wesley signature series. Addison-Wesley, 2003.

[33] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional Computing Series. Pearson Education, 1994.

[34] Jason Cheatham. *Snapshot Testing: Benefits and Drawbacks*. https://www.sitepen.com/blog/snapshot-testing-benefits-and-drawbacks.

[35] R.C. Gonzalez and R.E. Woods. *Digital Image Processing*. Pearson Education, 2011.

[36] Pradip K. Sinha. Image acquisition and preprocessing for machine vision systems. *Journal Name*, 2012.

[37] Richard Szeliski. *Computer Vision: Algorithms and Applications*. Springer, 1st edition, 2010.

[38] A. Géron. *Hands-on Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, 2017.

[39] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. Adaptive Computation and Machine Learning series. MIT Press, 2016.

[40] Paul Mooney. Detect retina damage from oct images. https://www.kaggle.com/code/paultimothymooney/detect-retina-damage-from-oct-images, 2024. Accessed: 2024-06-12.

[41] M. Kennedy and M. Harrison. *Effective PyCharm: Learn the PyCharm IDE with a Hands-on Approach*. Treading on Python Series. Amazon Digital Services LLC - KDP Print US, 2019.

[42] P. Barry. *Head First Python: A Brain-Friendly Guide*. Head first series. O'Reilly Media, 2016.

[43] S. Chacon. *Pro Git*. Expert's voice in software development. Apress, 2009.

[44] Wes McKinney. *Python for Data Analysis*. O'Reilly Media, 2017.

[45] B. Lubanovic. *FastAPI*. O'Reilly Media, 2023.

[46] M. Krebbs. *Deep Learning with Python*. Data Sciences Series. CreateSpace Independent Publishing Platform, 2018.

[47] B. Okken. *Python Testing with pytest: Simple, Rapid, Effective, and Scalable*. Pragmatic Bookshelf, 2017.

[48] Eli Stevens, Luca Antiga, and Thomas Viehmann. *Deep Learning with PyTorch*. Manning Publications, 2020.

Acronyms

AI	Artificial Intelligence
AMD	Age-related Macular Degeneration
ANN	Artificial Neural Networks
CNV	Choroidal NeoVascular membrane
CMYK	Cyan-Magneta-Yellow-Black
CNN	Convolutional Neural Network
DIP	Dependency Inversion Principle
DME	Diabetic Macular Edema
EDA	Exploratory Data Analysis
FCN	Fully Convolutional Networks
HSV	Hue-Saturation-Value
IoU	Intersection over Union
IRF	IntraRetinal Fluid
ISP	Interface Segregation Principle
KNN	K-Nearest Neighbours
LIP	Liskov Substitution Principle
MSE	Mean Square Error
MVP	Minimum Viable Product
OCP	Open-Closed Principle
OCT	Optical Coherence Tomography
PCA	Principal Component Analysis
PED	Pigment Epithelial Detachment
RGB	Red-Green-Blue
RVO	Retinal Vein Occlusion
SRF	SubRetinal Fluid
SRP	Single Responsibility Principle
SSIM	Structural Similarity Index Measure
TDD	Test-Driven Development
XP	eXtreme Programming
YOLO	You Only Look Once

Appendix A: Tools and Software

Pycharm

PyCharm is an integrated development environment (IDE) designed for Python programming. It offers advanced features like code analysis, debugging, and intelligent code completion to enhance developer productivity [41].

Python

Python is a high-level programming language that has a clear and readable syntax. It highlights simplicity and efficiency as key aspects., making it suitable for a wide range of applications, from web development and data analysis to artificial intelligence and automation [42].

Git

Git is a version control system used to track changes in source code during software development. It simplifies collaboration among multiple developers working on projects and provides efficient features like managing versions and branching [43].

Numpy

NumPy is a fundamental Python library for numerical computing, providing the creation and manipulation of large arrays and matrices efficiently. It offers a wide collection of mathematical functions, making it useful for data analysis and scientific computing tasks [44].

Pydantic

Pydantic is a Python library for data validation and settings management. It allows you to define data models using Python classes with type annotations, and it automatically validates and parses input data against these models. It is useful for reducing the risk of errors and making code more robust [45].

Sci-kit image

scikit-image is a Python library that provides a wide range of algorithms and tools for image processing and computer vision tasks. It integrates with other scientific Python libraries like NumPy, SciPy, and matplotlib [46].

Open-cv

OpenCV is an open-source library used for image processing and computer vision. It provides a range of algorithms and tools to manipulate and analyze numerical images and helps developers to easily implement operations like object detection, image enhancement, and pattern recognition [46].

Pytest

Pytest is a widely used testing framework for Python that simplifies writing and running unit tests. It provides a flexible and easy-to-use platform for organizing test cases, fixtures, and assertions, allowing developers to write concise and readable test code [47].

Covrage.py

coverage.py is a Python tool for measuring code coverage during testing. It tracks which lines of code are executed and reports the percentage of code covered by tests. It helps developers to identify untested areas to improve test suites [47].

Pytest-mock

pytest-mock is a plugin for the Python pytest testing framework, which provides support for mocking and stubbing functionalities during unit testing. It simplifies the process of mocking objects, functions, and methods [47].

Black

Black is an open-source code formatted for the Python programming language. It automatically reformats Python code to ensure consistent style and formatting, following the PEP 8 style guide by default [42].

Google Collab

Google Colab, short for "Collaboratory," is a free, cloud-based platform provided by Google that allows users to write and execute Python code in Jupyter Notebooks. It is particularly popular for data analysis, machine learning, and deep learning tasks because it provides free access to computational resources such as GPUs and TPUs. [48]

Trello

Trello is a project management tool that provides a kanban board and facilitates collaboration with the team members through multiple tools.

Overleaf

Overleaf is an online platform and collaborative LaTeX editor designed specifically for writing and editing documents using LaTeX, a typesetting system commonly used for academic and technical documents

TensorFlow

TensorFlow is an open-source software library developed by Google primarily for machine learning and deep learning applications. It enables developers to build and deploy machine learning models, particularly neural networks [38].

Sci-Kit Learn

scikit-learn (often abbreviated as sklearn) is a widely used open-source machine-learning library for Python. It is built on top of NumPy, SciPy, and matplotlib, and provides a simple and efficient set of tools for data mining and data analysis [38].