BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INTELLIGENT SYSTEMS
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

GENERATION OF SYNTHETIC RETINAL IMAGES IN HIGH RESOLUTION

GENEROVÁNÍ SYNTETICKÝCH SNÍMKŮ SÍTNICE VE VYSOKÉM ROZLIŠENÍ

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR **Bc. TOMÁŠ AUBRECHT**
AUTOR PRÁCE

SUPERVISOR **Prof. Ing., Dipl.-Ing. MARTIN DRAHANSKÝ, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2020

Brno University of Technology
Faculty of Information Technology

Department of Intelligent Systems (DITS) Academic year 2019/2020

Master's Thesis Specification

21968

Student:	**Aubrecht Tomáš, Bc.**
Programme:	Information Technology Field of study: Information Systems
Title:	**Generation of Synthetic Retinal Images with High Resolution**
Category:	Computer Graphics

Assignment:

1. Study the literature in the area of processing and recognition of human retinal images, especially in ophthalmology sources, incl. generation of synthetic retinal images.
2. Propose your own method for generation of synthetic retinal images in high resolution (over 12 Mpix).
3. Implement the proposed method from the previous point. Generate a database of at least 1,000 images without pathological damage.
4. Test the implemented solution from the previous point, compare your results with real retinal images (e.g. vessels distribution) and summarize the achieved results.

Recommended literature:

- BONALDI, Lorenza, et al. Automatic generation of synthetic retinal fundus images: vascular network. *Procedia Computer Science*, 2016, 90: 54-60.
- WONG, Tien Yin; TING, Daniel Shu Wei. Generative Adversarial Networks (GANs) for Retinal Fundus Image Synthesis. In: *Computer Vision-ACCV 2018 Workshops: 14th Asian Conference on Computer Vision, Perth, Australia, December 2-6, 2018, Revised Selected Papers*. Springer, 2019. p. 289.
- FIORINI, Samuele, et al. Automatic Generation of Synthetic Retinal Fundus Images. In: *Eurographics Italian Chapter Conference*. 2014. p. 41-44.

Requirements for the semestral defence:

- Items 1 and 2.

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor:	**Drahanský Martin, prof. Ing., Dipl.-Ing., Ph.D.**
Consultant:	Biswas Sangeeta, Ph.D., UITS FIT VUT
Head of Department:	Hanáček Petr, doc. Dr. Ing.
Beginning of work:	November 1, 2019
Submission deadline:	June 3, 2020
Approval date:	October 31, 2019

Abstract

Special equipment, a fundus camera, is needed to capture the retina, which is the most important part of the human eye. Therefore, the main objective of this work is to design and implement a system that would be able to generate retinal images. The proposed solution uses an image-to-image translation, where the system is provided with a black and white image at the input containing only bloodstream, on the basis of which a color image of the entire retina is generated. The system consists of two neural networks: a generator, which generates retinal images, and a discriminator, which classifies these images as real or synthetic. Training of this system was performed on 141 images from publicly available databases. A new database was created with more than 2,800 images of healthy retinas in a resolution of 1024×1024. This database could be used as a learning tool for ophthalmologists or for the development of various applications working with retinas.

Abstrakt

K pořízení snímků sítnice, která představuje nejdůležitější část lidského oka, je potřeba speciálního vybavení, kterým je fundus kamera. Z tohoto důvodu je cílem této práce navrhnout a implementovat systém, který bude schopný generovat takovéto snímky bez použítí této kamery. Navržený systém využívá mapování vstupního černobílého snímku krevního řečiště sítnice na barevný výstupní snímek celé sítnice. Systém se skládá ze dvou neuronových sítí: generátoru, který generuje snímky sítnic, a diskriminátoru, který klasifikuje dané snímky jako reálné či syntetické. Tento systém byl natrénován na 141 snímcích z veřejně dostupných databází. Následně byla vytvořena nová databáze obsahující více než 2,800 snímků zdravých sítnic v rozlišení 1024×1024. Tato databáze může být použita jako učební pomůcka pro oční lékaře nebo může poskytovat základ pro vývoj různých aplikací pracujících se sítnicemi.

Keywords

human eye, eye retina, synthetic retinal images, image processing, image generation, machine learning, neural networks, GAN, high resolution, Python, TensorFlow

Klíčová slova

lidské oko, sítnice oka, syntetické snímky sítnice, zpracování obrazu, generování obrazu, strojové učení, neuronové sítě, GAN, vysoké rozlišení, Python, TensorFlow

Reference

AUBRECHT, Tomáš. *Generation of Synthetic Retinal Images in High Resolution.* Brno, 2020. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Prof. Ing., Dipl.-Ing. Martin Drahanský, Ph.D.

Rozšířený abstrakt

Diplomová práce se zabývá generováním syntetických snímků sítnic ve vysokém rozlišení. Lidské oko je párový orgán, který nám poskytuje zrak. Díky němu můžeme vnímat okolní svět a orientovat se v prostoru. Nejdůležitější část lidského oka tvoří právě sítnice, která je zároveň i tou nejcitlivější částí. Z tohoto důvodu mohou různé nemoci nebo sebemenší poškození sítnice vést ke ztrátě zraku. Je tedy důležité, aby si člověk svůj zrak chránil, protože jeho ztráta vede k významnému zhoršení kvality života. K pořízení snímků sítnice je potřeba speciálního vybavení (fundus kamery), proto není jednoduché získat takovéto snímky ve větším počtu. Z tohoto důvodu je cílem této práce navrhnout a implementovat systém, který bude schopný generovat nové syntetické snímky sítnic ve vysokém rozlišení, které budou nerozeznatelné od těch reálných. Dalším krokem je pomocí tohoto systému vytvoření nové databáze snímků zdravých sítnic, tedy sítnic bez patologických nálezů.

Teoretická část se zaměřuje na anatomii lidského oka, která je důležitým základem pro pochopení jeho činnosti. V této části je popsán i způsob vyšetření očního pozadí a následně jsou uvedena vybraná onemocnění sítnice spolu s jejich příznaky. Na základě těchto informací je člověk schopný si vytvořit představu o tom, jak vypadá zdravá sítnice. Z technického hlediska je pozornost věnována základním typům strojovému učení a více se zaměřuje na neuronové sítě, pomocí kterých byl realizován navržený systém pro generování snímků sítnic. Konkrétněji se zabývá speciálními typy neuronových sítí, jako jsou konvoluční a generativní neuronové sítě. Praktická část této práce poskytuje detailní popis návrhu a následné implementace daného systému. V poslední části je uveden proces učení tohoto systému spolu se zhodnocením dosažených výsledků.

Navržené řešení využívá principu mapování vstupního snímku na výstupní. Na vstupu systému je černobílý obrázek obsahující krevní řečiště sítnice, na jehož základě se vygeneruje barevný snímek celé sítnice. Samotný systém je tvořen generativní kompetitivní sítí. Ta se skládá ze dvou dílčích neuronových sítí, kde jednou z nich je generátor, který ze vstupního obrázku generuje snímky sítnic, a druhou je diskriminátor, který provádí klasifikaci, zdali jsou dané snímky reálné či syntetické. Diskriminátor má na svém vstupu dva snímky. Prvním z nich je černobílý snímek krevního řečiště nějaké sítnice a druhým je snímek odpovídající sítnice, který je následně posouzen.

Aby byl tento systém schopen vytvářet realisticky vypadající snímky sítnic, musí se to nejprve naučit. Samotné učení probíhalo na snímcích z několika veřejně dostupných databází sítnic, které obsahují i potřebné snímky krevních řečišť. Tyto databáze dohromady poskytly 141 snímků. Generátor a diskriminátor byly učeni současně, kde cílem bylo, aby generátor vytvářel snímky v takové kvalitě, aby diskriminátor nebyl schopný rozlišit, zdali se jedná o reálné či syntetické snímky. Zároveň cílem diskriminátoru bylo, aby jeho rozlišovací schopnost dosáhla co nejvyšší úrovně. Při tomto současném učení bylo potřeba dávat pozor na to, aby jedna z těchto sítí nedominovala té druhé, neboť systém jako celek by následně produkoval výsledky nízké kvality. Z tohoto důvodu bylo potřeba najít rovnováhu mezi těmito sítěmi.

Po naučení systému již nebylo potřeba diskriminátoru a dále se pracovalo pouze s naučeným generátorem. Pomocí tohoto generátoru byla vytvořena databáze, která obsahuje víc než 2,800 snímků zdravých sítnic, které jsou v rozlišení 1024×1024 pixelů. Tato databáze může být následně použita jako učební pomůcka pro oční lékaře nebo může poskytovat základ pro vývoj různých aplikací pracujících se sítnicemi. Může se jednat například o aplikace pohybující se v oblasti medicínských nebo biometrických systémů. Mnohé z těchto vytvořených snímků jsou nerozeznatelné od snímků skutečných sítnic, což bylo cílem této práce.

Generation of Synthetic Retinal Images in High Resolution

Declaration

I hereby declare that this Master's thesis was prepared as an original work by the author under the supervision of prof. Ing., Dipl.-Ing. Martin Drahanský, Ph.D. The supplementary information was provided by Biswas Sangeeta, Ph.D. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

. . . .
Tomáš Aubrecht
June 1, 2020

Acknowledgements

I would like to thank Mr. Martin Drahanský for his thorough guidance and his hints, and Miss Biswas Sangeeta for her help during the thesis work, her valuable advice and her time spent on consultations.

Contents

Chapter 1

Introduction

Eyesight allows us to interpret the surrounding environment using light in its visible spectrum. Thanks to this, we can perceive contrast, contours of objects and their distance from us. It also contributes to the perception of spatial orientation. For this reason, it is important to protect our eyesight, as its loss leads to a significant deterioration in the quality of life.

We begin to see when the cornea, together with the lens of the eye, focuses light from our surroundings on the light-sensitive membrane at the back of the eye, which is called the retina. It contains specialized light-sensitive cells: rods that allow the perception of contrast, and cones that allow the perception of color. These cells convert the light into electrical signals that are transmitted to the visual cortex of the brain by the optic nerve. Therefore, the retina is the most sensitive and most important part of the human eye, and diseases or the slightest mechanical damage can lead to loss of vision.

Machine learning is an application of artificial intelligence that provides systems with the ability to automatically learn and improve from previous experience without being explicitly programmed. One of the most popular areas of machine learning today is deep learning. This has been inspired by the human brain, and it generally consists of a large number of parameters with multiple nonlinear layers. Generative models are an example of deep learning, more specifically, generative adversarial networks.

A generative adversarial network (GAN) is a type of neural network that is based on two models: a generator and a discriminator. The generator produces a synthetic image from random noise, and the discriminator predicts whether the image is real or created by the generator. The generator is trained to be able to fool the discriminator to such an extent that it is not possible for the discriminator to distinguish between real and fake images. Meanwhile, the discriminator constantly adapts to the gradually improving capabilities of the generator. Therefore, both models are trained to surpass the other.

Synthesizing realistic images of the eye fundus is a challenging task. Recent advances in technology have brought high computational power, leading machine learning to neural networks with deep architectures. Considering advances in deep learning algorithms, GAN provides a valuable framework. Rapid enhancement of GANs facilitated the synthesis of realistic-looking images, leading to slightly anatomically consistent retinal images with reasonable visual quality [50].

1.1 Aims

The aim of this thesis is to design and implement an algorithm that allows the automatic generation of high-resolution digital images of the retina using the generative adversarial network. In the next step, this network needs to be trained using real retinal images from existing databases. The results obtained from the algorithm will be compared with these real images, and in case of high accuracy of this algorithm, a database of synthetic retinal images will be created. This database could be used in practice for the development of various medical or biometric systems.

1.2 Contents

Chapter 2 focuses on the anatomy of the human eye, which is an important basis for understanding its physiology and the risks posed by various diseases. This chapter also describes the most common methods of examining the eye and individual eye diseases, along with a description of their symptoms and possible treatments. Information about the human eye was taken from my previous work [8]. Chapter 3 provides an introduction to the machine learning on which the proposed algorithm is based. This introduction includes types of machine learning along with a description of the neural network. Chapter 4 contains the proposed solution of the system for generating synthetic images of the retina. Its implementation is given in Chapter 5. The actual training and testing of the proposed system are described in Chapter 6. The final chapter, Chapter 7, contains a summary of this thesis, including the final evaluation of achieved results and plans for future work.

Chapter 2

Human Eye

Human eyes are paired organs of the visual system, which provide us with vision, an ability to perceive the surrounding world and to orient ourselves in space thanks to the light in its visible spectrum reflected by objects in the environment. Up to 80 % [23] of information from the external environment is perceived by sight. Therefore, the eye is the most important sensory organ. It has an approximately spherical shape, and it is made up of three layers, enclosing various anatomical structures. The outermost layer is composed of the cornea and sclera. The middle layer consists of the choroid, ciliary body, pigmented epithelium and iris, and the innermost layer is the retina.

2.1 Vision

When looking at an object, light rays reflect from that object and enter the cornea. The light rays are refracted and concentrated in one place through the cornea, lens and vitreous humor. Of these three structures, only the lens can change its optical power, thus ensuring that the rays are concentrated on the point of sharpest vision. The resulting image on the retina is turned upside down. Photons of light falling on the light-sensitive cells of the retina are converted into electrical signals that are transmitted to the brain by the optic nerve. These signals are interpreted as the resulting image in the visual cortex of the brain [23].

2.2 Anatomy

The human eye is a very complex system made up of many parts that must work together perfectly. The most important parts are described below. They are shown in Figure 2.1.

- **The cornea** is a transparent dome-shaped layer covering the anterior portion of the eyeball. The cornea, with regard to its optical power, is the most important component of the optical system of the eye, and is the largest contributor to quality vision. Its main function is to refract light. It is responsible for focusing most of the light that enters the eye. The cornea is colorless, completely transparent, and without blood vessels, which may prevent it from refracting light properly and may adversely affect vision. Since there are no nutrient-supplying blood vessels in the cornea, tears and the aqueous humor in the anterior chamber provide the cornea with nutrients. It represents a mechanical and chemically impermeable barrier between the inner and outer environment together with the conjunctiva, sclera and tear film.

- **The conjunctiva** is the clear, thin membrane that consists of two segments: bulbar conjunctiva, which covers the anterior part of the sclera, and palpebral conjunctiva, which covers the inner surface of both the upper and lower eyelids. The conjunctiva has many small blood vessels that provide nutrients to the eye and lids. Its main function is to keep the eye moist and lubricated by producing mucus and tears. It also contributes to the protection from dust, debris and microorganisms that can cause an infection.

- **The sclera**, also known as the white of the eye, is the protective, opaque, outer layer of the human eye. The whole sclera is white, contrasting with the coloured iris. It is continuous with the cornea offering resistance to internal and external forces to protect sensitive eye structures stored inside. The sclera also provides a sturdy attachment for the extraocular muscles that control the movement of the eyes. It is perforated by many nerves and vessels passing through its posterior part, where the hole is formed by the optic nerve.

- **The choroid**, also known as the choroidea, is another layer surrounding the eyeball that lies between the sclera and the retina. It provides oxygen and nourishment to the outer layers of the retina and maintains the temperature and volume of the eye.

- **The anterior chamber** of the eyeball is the space inside the eye that is behind the cornea and in front of the iris. It is filled with a clear, watery fluid known as the aqueous humor. This is where the excess fluid can normally flow out. If the normal outflow of aqueous humour is blocked, the intraocular pressure is increased and glaucoma usually develops. This can lead to progressive damage to the optic nerve head, and eventually blindness.

- **The iris** is a thin, circular structure located behind the anterior chamber that is usually strongly pigmented. The color of our eyes is determined by the amount of pigment in the iris. It contains a circular opening in the center called a pupil. The primary function of the iris is to regulate the amount of light entering the eye by dilating or contracting the pupil. The iris contracts the pupil when the ambient illumination is high and dilates it when the illumination is low [26].

- **The lens** is composed of transparent, flexible tissue, and is located directly behind the iris and the pupil. It is important for the refraction of light and its accommodation. The accommodation is a process of changing the curvature of the lens, allowing closer objects to be brought into better focus by changing the optical power of the lens.

- **The posterior chamber** of the eyeball is the second chamber consisting of small space directly behind the iris and in front of the lens. Like the anterior chamber of the eye, it is also filled with the aqueous humor. This fluid normally passes into the posterior chamber from where it flows into the anterior chamber. There, the excess fluid can flow out of the eye.

- **The vitreous humor**, also known simply as the vitreous, is a clear, colorless fluid that fills the space behind the lens and in front of the retina in the eye. It has a firm gelatinous consistency, and it makes up most of the volume of the eyeball. The vitreous helps to hold the shape of the eye, and its pressure helps to keep the retina in place.

- **The optic nerve** connects the eye to the visual cortex of the brain. It is the nerve that transmits visual information in the form of impulses formed by the retina. These impulses are dispatched through the optic nerve to the brain, which interprets them as images. Glaucoma is a disease which results in damage to the optic nerve and causes vision loss. It is caused by high intraocular pressure, which compresses the optic nerve and causes its cells to die. It is referred to as the atrophy of the optic nerve.

- **Retina** is the most important part of this work, so it is described separately and in more detail in the following Section 2.3.

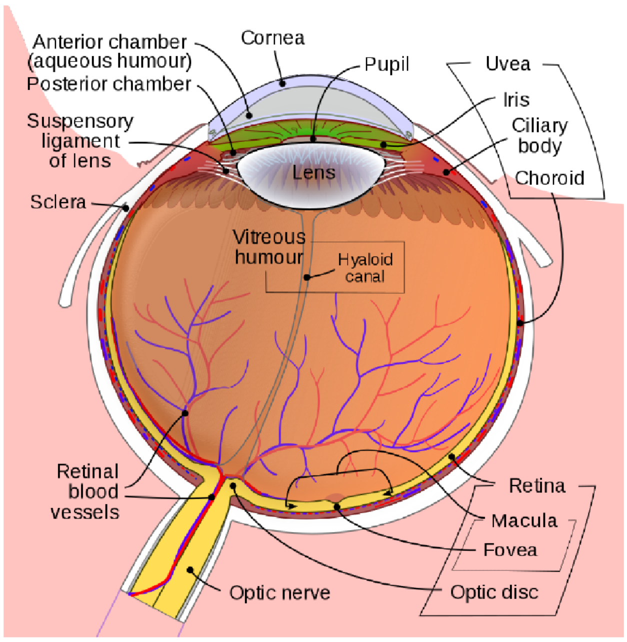


Figure 2.1: Schematic diagram of the human eye [43].

2.3 Retina

The retina is the most important and also the most sensitive part of our eye. It is a thin layer of tissue that lines the inner surface of the back of the eyeball. The retina processes light through a layer of light-sensitive cells, responsible for detecting qualities such as color and light intensity. These specialized cells are called photoreceptors. The retina captures the light falling on these photoreceptors and converts the light into neural signals that are transmitted through the optic nerve to the visual cortex of the brain for visual recognition.

Photoreceptors

A photoreceptor is a specialized light-sensitive cell found in the retina that is responsible for converting light into signals that can stimulate biological processes. The photoreceptor absorbs photons that are striking the retina, which triggers a change in the membrane potential of the cell. There are two types of photoreceptor cells in the human retina: rods and cones. There are major functional differences between the rods and cones. Rod cells are much more sensitive than cone cells. At very low light levels, the visual experience is based solely on the rod signal, so they are responsible for night vision. However, they do not mediate color vision, which is the main reason why colors are much less apparent in dim light, and not at all at night. The rods are concentrated at the outer edges of the retina, and are used in peripheral vision. Cones require significantly larger number of photons to produce a signal. They are responsible for the perception of color and for high spatial acuity used for tasks such as reading. Cones are most concentrated in the center of the retina in an area called the macula, and their density gradually decreases towards the outer edges of the retina [29].

Macula

The macula is a yellow oval-shaped area near the center of the retina where the light is focused by the cornea and lens. The macula is responsible for the central, high-resolution and color vision. Therefore, the macula provides us with the ability to read and see in great detail. In the very center of the macular region is the fovea that has a very high concentration of photoreceptor cells, more specifically, a high density of cones and low density of rods.

Optic Disc

The optic disc, also called the optic nerve head, is located at the optic papilla, where the optic nerve fibres leave the eye. There are no photoreceptors in this area, so it is sometimes called the blind spot. The optic disc appears as an approximately oval area, and it is the entry point for the blood vessels that supply the retina. These structures can be seen in Figure 2.2.

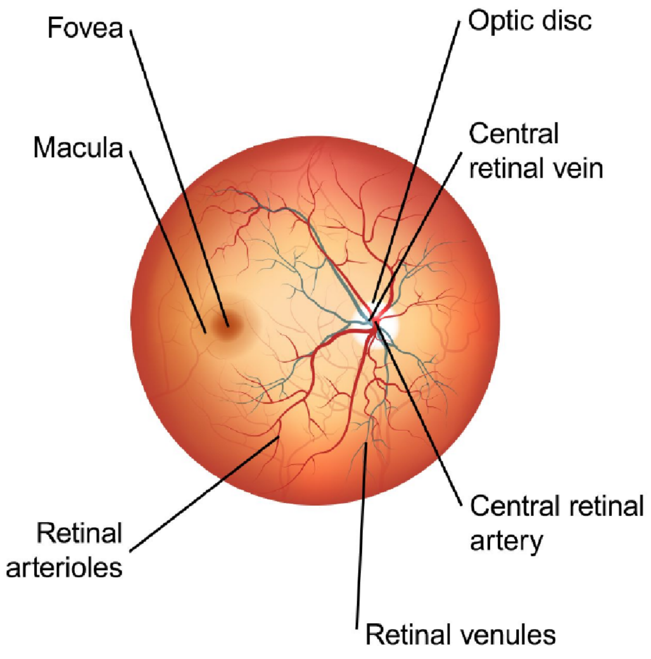


Figure 2.2: Retina of the human eye [49].

2.4 Eye Examination

Ophthalmology is a branch of medicine dealing with anatomy and physiology of the eye, and with the diagnosis, treatment and prevention of diseases of the whole visual system. This is a very specialized field, especially since the eye is a very complicated apparatus. An ophthalmologist is a medical doctor who specializes in diagnosing and treating eye-related conditions. In other words, an ophthalmologist is a specialist in ophthalmology. An eye examination is a series of tests performed by an ophthalmologist, evaluating vision and ability to focus on and recognize objects.

Ophthalmoscopy

Ophthalmoscopy is an examination of the back part of the eye. This part of the eye is called the fundus, and consists of: retina, optic disc, choroid and blood vessels. Ophthalmoscopy may also be called funduscopy or retinal examination. Through ophthalmoscopy, an eye doctor can find evidence of many kinds of eye problems including, but not limited to, glaucoma, high blood pressure damage, retinal detachment, diabetes, eye tumors, and many other problems. The dilation of the pupils, also known as mydriasis, is a simple and effective way to better observe the structures behind them. This is often done with eye drops before the examination. There are three different types of examinations: direct, indirect and slit-lamp examination.

Direct ophthalmoscopy produces an upright image of approximately 15× magnification. The handheld instrument that our primary care physician uses to look into our eyes is called a direct ophthalmoscope. One can be seen in Figure 2.3. It is about the size of a small flashlight, and it consists of a concave mirror and a battery-powered light. The doctor looks through a single monocular eyepiece into the eye of a patient in a darkened room. The ophthalmoscope is equipped with a rotating disc of lenses to permit the eye to be examined at different depths and magnifications. It provides good, but limited visualization of the back of the eye. This type of ophthalmoscope is most commonly used during a routine physical examination.

Indirect ophthalmoscopy provides a wider view of the inside of the eye and produces an inverted image of 2 to 5× magnification using an indirect ophthalmoscope (Figure 2.4). An indirect ophthalmoscope can be either monocular or binocular. It constitutes a bright light attached to a headband positioned on the forehead of the eye doctor and magnifying lenses. The eye doctor holds the eye open while shining a very bright light into the eye using this indirect ophthalmoscope and views the back of it through the lens held close to the eye. Some pressure may be applied to the eye using a small, blunt probe. The pupil must be fully dilated for a satisfactory result. This examination is usually used for peripheral viewing of the retina, and to look for a detached retina.

The slit lamp is the most widely used ophthalmic device. It has a place for us to rest our chin and forehead. This will help keep our head steady. This procedure gives us the same view of the eye as an indirect examination, but with greater magnification. A microscope is connected to a lamp, which is a high-intensity light source that can be focused to shine a thin ray of light into the eye. The doctor directs the light right into the eye of the patient, thus illuminating the area accurately. During the examination, the tissues are illuminated either by a thin ray of light, or by reflected light. By examining the illuminated eye with the microscope, the ophthalmologist then obtains a magnified image of the observed area, allowing the detection of very subtle changes and symptoms of eye diseases.

Figure 2.3: Direct ophthalmoscope [34].

Figure 2.4: Indirect ophthalmoscope [46].

Fundus Photography

Fundus photography uses a fundus camera to record images of the condition of the interior surface of the eye, also known as the fundus. Ophthalmologists use these retinal photographs for detailed evaluation as well as to document clinical observations and possible diagnosis of eye diseases. The fundus camera (Figure 2.5) is a device that replaces the ophthalmoscope. It is a specialized low power microscope with an attached camera, and it is based on the principle of monocular indirect ophthalmoscopy. The optics of a fundus camera are similar to those of an indirect ophthalmoscope in that the observation and illumination systems follow dissimilar paths. Fundus cameras are described by the angle of view, and provide an upright, magnified view of the back of an eye. A typical camera captures images between 30° and 50° of the retinal area with a magnification of 2.5×. This relation can be modified using zoom or auxiliary lenses. Wide-angle fundus cameras capture images between 45° and 140°, and provide proportionately less retinal magnification. For a better inspection, dilating eye drops are applied prior to the examination to enlarge the pupil, thus increasing the angle of observation [9].

Figure 2.5: Fundus camera [38].

2.5 Retinal Diseases

Retinal diseases vary widely, but most of them cause visual symptoms. Retinal diseases can affect any part of the retina, and they are always very serious, often irreversible and can lead to severe vision loss or blindness. Treatment is available only for some retinal diseases. Depending on the retina condition, treatment goals may be to stop or slow the disease and preserve, improve or restore the vision. Common retinal diseases and conditions are described below.

Macular Degeneration

Macular degeneration, also known as age-related macular degeneration (AMD or ARMD), is a macular disease that occurs in patients over age 50, and is the most common cause of practical blindness in economically developed countries. With the increasing number of seniors, it becomes a major societal health problem. Several factors influence the origin and development of this disease. In addition to increasing age, it can also be high blood pressure, smoking, poor eating habits and the associated obesity and genetic predisposition. Patients describe its symptoms so that visual acuity gradually decreases, they are complaining about image distortions, and in more advanced stages, a blurred or sometimes even black spot appears in the center of the field of view. Color vision also deteriorates. There is currently no known cure for macular degeneration, but there are options to reduce the risk and possibly slow the progression of the wet form. Vision will no longer improve and only the current quality of vision will stabilize [24].

AMD is divided into 2 forms: dry (atrophic, nonexudative) and wet (exudative). Up to 90 % of patients are affected by the dry form, but it causes severe visual damage in only 12—21 %. Fewer patients suffer from the wet form, but it is far more dangerous than the dry form because, it progresses very quickly. Both forms can be combined during disease. In the macular area of the patients, changes and loss of retinal pigment epithelium and drusen are found. Drusen are divided according to their appearance and size into hard and soft. Their comparison can be seen in Figure 2.6 and 2.7. Hard drusen are small bounded deposits of yellowish color under the retina. On the contrary, soft drusen have no sharp boundaries and may even coalesce, they are associated with a significantly higher risk of the formation of the wet form of AMD [28].

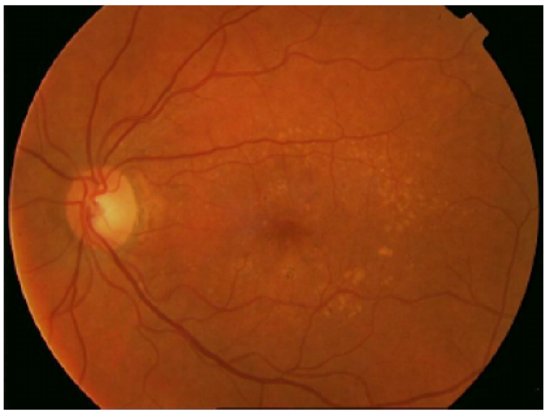


Figure 2.6: Hard drusen [28].

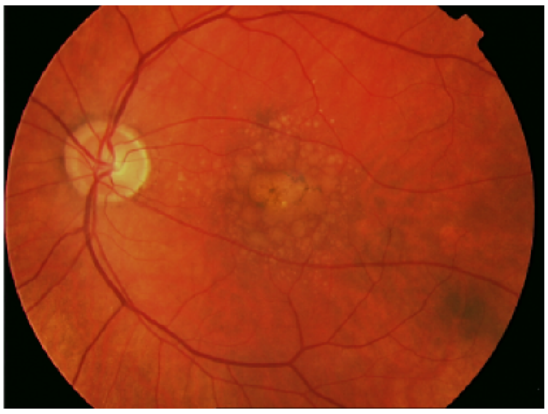


Figure 2.7: Soft drusen [28].

Dry AMD starts with the build-up of drusen in the retina. Vision is usually good or only slightly reduced at this stage. Most of these patients with mild dry AMD can continue to read and drive, although it may not be as easy as it was when they were younger. Some patients, but not all, progress to a more advanced stage of dry AMD called geographic atrophy (Figure 2.8). This can result in severe loss of central vision and loss of the ability to read and drive. Even in these severe cases, patients almost always retain normal peripheral vision, enough to see where they are going. Unfortunately, there is no treatment for dry AMD. However, supplementation of antioxidant vitamins C, E and minerals zinc, selenium and essential omega-3 fatty acids may have a beneficial effect on preventing or slowing its progression. A diet rich in fish, vegetables and fruits also has a supporting role [28].

Wet AMD occurs when abnormal new blood vessels grow into the retina and start leaking fluid. Macular edema is the build-up of this fluid in the macula. This causes the retina to swell, and the longer it is swollen, the more the retinal fibres deteriorate. Because these blood vessels are abnormal, they are more fragile than typical blood vessels and can bleed into the retina. This bleeding can cause irreversible damage to the photoreceptors and rapid vision loss if left untreated. A characteristic image of a retina with the macular edema can be seen in Figure 2.9. It is usually, but not always, preceded by the dry form of AMD. The wet form progresses faster compared to the dry form, and the loss of vision is more significant. Rapid deterioration occurs within a few weeks and practical blindness within a few months. Treatment previously consisted of destruction of the neovascular membrane by photocoagulation or thermotherapeutic laser. However, treatment results were variable. The starting point should be a more targeted so-called photodynamic therapy, in which the intravenously injected substance is absorbed by the target tissue and then activated by laser [28].

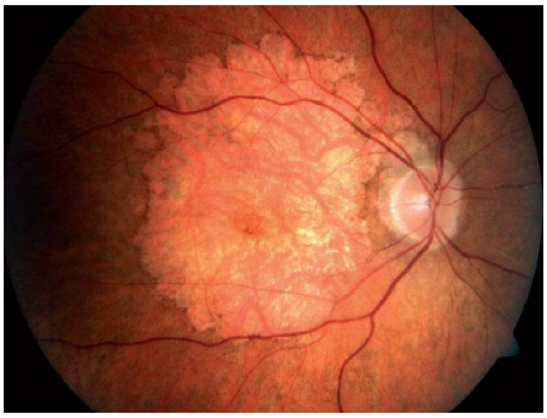


Figure 2.8: Geographic atrophy, which is a more advanced stage of dry AMD [35].

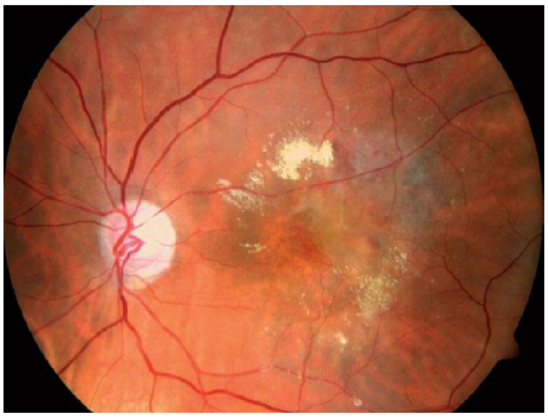


Figure 2.9: Wet form of AMD with the macular edema [36].

Diabetic Retinopathy

Diabetic retinopathy is a diabetes complication that affects eyes. Retinopathy occurs when high blood sugar levels lead to the blockage of the tiny blood vessels that nourish the retina, cutting off its blood supply. The weakened blood vessels leak fluid into the retina and some of them break and bleed. This is called retinal haemorrhage, and can be seen in Figure 2.10. As the disease becomes more advanced, new abnormal blood vessels may grow and these new blood vessels can bleed, cause cloudy vision, and destroy the retina. This condition can develop in anyone who has type 1 or type 2 diabetes. The longer the patient has diabetes and the less controlled his blood sugar is, the more likely he is to develop this eye complication. Diabetic retinopathy begins before the patient has any symptoms, but as the problem gets worse the patient may have: blurred vision, temporary or permanent blindness or distortion of vision. Early treatment is the key to reduce vision loss. A laser is used to seal leaking blood vessels or destroy abnormal blood vessels [42].

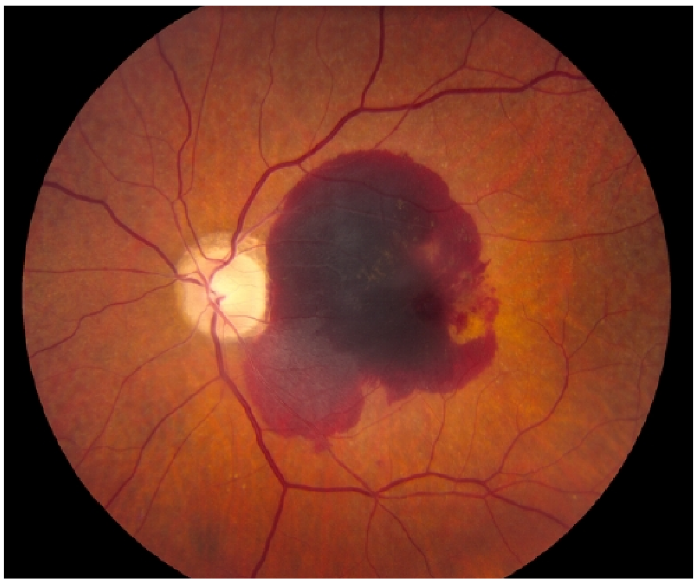


Figure 2.10: Retinal haemorrhage [11].

Retinal Detachment

A retinal detachment is defined by the presence of fluid under the retina. If a hole develops in the retina, then the suction force is lost and the fluid that normally fills the inside of the eye passes through the hole and enters the space underneath the retina. As more fluid passes underneath it, the retina gradually detaches from the inner wall of the eye. If the retina remains detached, it will slowly deteriorate and lose function permanently, but if the retina can be reattached with surgery quickly enough, it is possible to recover some function and to avoid permanent vision loss [42].

Retinal Vein Occlusion

A retinal vein occlusion is a blockage of one of the veins draining blood from the eye. Retinal vein occlusion is divided into categories, based on the size of the vein which is blocked. A branch retinal vein occlusion is a blockage of one branch only, and affects only part of the retina and a central retinal vein occlusion is a blockage of the main vein and affects the whole retina. If there is a very severe blockage and the blood flow stops altogether, the retinal cells die due to lack of oxygen. This is called ischaemia, and there is no treatment that can bring the cells back to life. The increased pressure in the small vessels in the eye results in fluid leaking into the retina, making it swollen. A swollen retina does not see as well, and the longer the retina remains swollen, the more the vision deteriorates with time. Possible treatment options are intravitreal injections to reduce the swelling, or laser surgery. If the blood supply is not restored, new blood vessels can grow into the retina. These new vessels are very fragile and can bleed, which can dramatically reduce the vision. In some cases, this bleeding will require surgery to remove the blood in order to restore vision [42].

Retinitis Pigmentosa

Retinitis pigmentosa is a group of rare, genetic disorders that involve a breakdown and loss of cells in the retina. The rods are more severely affected than cones in the early stages, and people have difficulty seeing at night and lose the peripheral vision. The loss of rods eventually leads to a breakdown and loss of cones. In the late stages, people tend to lose more of the visual field, developing tunnel vision. Retinitis pigmentosa is diagnosed by an examination of the retina, which typically reveals abnormal, dark pigment deposits that streak the retina. There is currently no cure for this disorder [37].

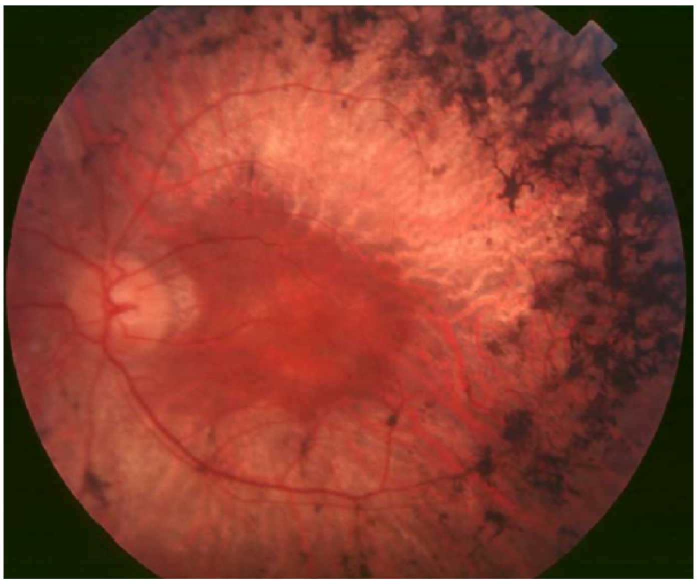


Figure 2.11: Fundus of a patient with retinitis pigmentosa [21].

Chapter 3

Machine Learning

To solve a problem on a computer, we need an algorithm. An algorithm is a sequence of instructions that should be carried out to transform the input to output. For some tasks, however, we do not have an algorithm. Therefore, we do not know how to transform the input to output. What we lack in knowledge, we make up for in data. With advances in computer technology, we currently have the ability to store and process large amounts of data, as well as to access it from physically distant locations over a computer network. There are certain patterns in the data. Such patterns may help us better understand the data, or we can use those patterns to make predictions. Assuming that the future, at least the near future, will not be much different from the past when the sample data was collected, the future predictions can also be expected to be right. Application of machine learning methods to large databases is called data mining. Its application areas are abundant. In finance, banks analyze their past data to build models to use in credit applications, fraud detection, and the stock market. In manufacturing, learning models are used for optimization, control, and troubleshooting. In medicine, learning programs are used for medical diagnosis. In telecommunications, call patterns are analyzed for network optimization and maximizing the quality of service, and in science, large amounts of data in physics, astronomy, and biology can only be analyzed fast enough by computers.

Machine learning is not just a database problem; it is also a part of artificial intelligence. To be intelligent, a system that is in a changing environment should have the ability to learn. The key concept is learning from data since data is what we have. Machine learning, then, is about making computers modify or adapt their actions, so that these actions get more accurate, where accuracy is measured by how well the chosen actions reflect the correct ones. If the system can learn and adapt to such changes, the system designer does not need to foresee and provide solutions for all possible situations. Machine learning also helps us find solutions to many problems in vision, speech recognition, and robotics. One example of pattern recognition is face recognition. This is a task we do effortlessly. Every day, we recognize family members and friends by looking at their faces or from their photographs, despite differences in the pose, lighting, hairstyle, and so forth. But we do it unconsciously and are unable to explain how we do it. Because we are not able to explain our expertise, we cannot write the computer program. At the same time, we know that a face image is not just a random collection of pixels. A face has structure. It is symmetric. There are the eyes, the nose, the mouth, located in certain places on the face. Each face of a person is a pattern composed of a particular combination of these. By analyzing sample face images of a person, a learning program captures the pattern specific to that person and then recognizes by checking for this pattern in a given image [1].

One of the most interesting features of machine learning is that it lies on the boundary of several academic disciplines, principally computer science, statistics, mathematics, and engineering. This has been a problem as well as an asset since these groups have traditionally not talked to each other very much [32]. Machine learning is programming computers to optimize a performance criterion using example data or past experience. We have a model defined up to some parameters, and learning is the execution of a computer program to optimize the parameters of the model using the training data or past experience. The model may be predictive to make predictions in the future, or descriptive to gain knowledge from data, or both. Machine learning uses the theory of statistics in building mathematical models, because the core task is making inference from a sample. The role of computer science is twofold. First, in training, we need efficient algorithms to solve the optimization problem, as well as to store and process the massive amount of data we generally have. Second, once a model is learned, its representation and algorithmic solution for inference needs to be efficient as well. Training does not happen very often, and is not usually time-critical, so it can take longer. However, we often want a decision about a test point quickly, and there are potentially lots of test points when an algorithm is in use, so this needs to have low computational cost. In certain applications, the efficiency of the learning or inference algorithm, namely, its space and time complexity, maybe as important as its predictive accuracy.

3.1 Types of Machine Learning

Learning can be loosely defined as a process of getting better at some task through practice. This leads to a couple of vital questions: how does the computer know whether it is getting better or not, and how does it know how to improve? There are several possible answers to these questions, and they produce different types of machine learning. Machine learning algorithms are typically classified into three broad categories: supervised learning, unsupervised learning, and reinforcement learning [52].

Supervised Learning

The most common type of learning is supervised learning. A training set of examples with the corresponding targets are provided, and based on this training set, the algorithm generalizes to respond correctly to all possible inputs. This is also called learning from examples [32]. When the target vectors are categorical, the problems are known as classification or pattern recognition, and when the target vectors are real-valued, the problems are known as regression.

If we had examples of every possible piece of input data, then we could put them together into a big look-up table, and there would be no need for machine learning at all. The thing that makes machine learning better is a generalization: the algorithm should produce sensible outputs for inputs that weren't encountered during learning. This also has the result that the algorithm can deal with noise, which are small inaccuracies in the data. In other words, the goal of supervised learning is to learn mapping from the input to an output whose correct values are provided by a supervisor.

This work is based purely on supervised learning, so further details are given in the following Section 3.2 on artificial neural networks and Section 3.3 on deep learning.

Unsupervised Learning

In unsupervised learning, there is no supervisor, no targets are defined so that the training data consist of only a set of input vectors. The goal is to find the regularities in the input data. There is a structure to the input space such that certain patterns occur more often than others, and we want to see what generally happens and what does not. In statistics, this is called density estimation. One method for density estimation is clustering. Therefore, a variety of clustering algorithms are canonical examples of unsupervised learning. One specific example of density-based clustering is shown in Figure 3.1 below.

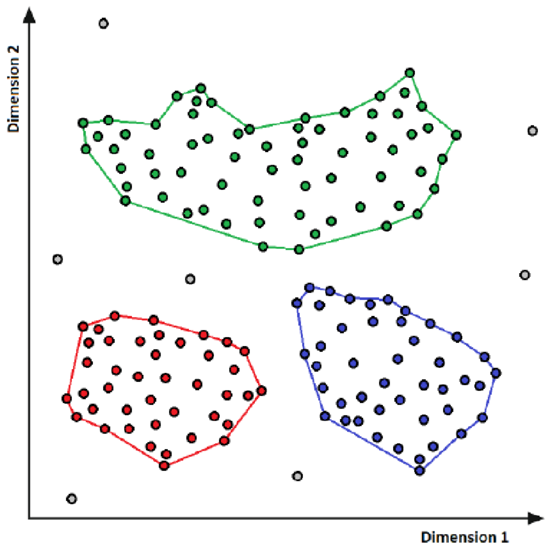


Figure 3.1: Example of density-based clustering that connects areas of high input data density into clusters.

Reinforcement Learning

This is somewhere between supervised and unsupervised learning. The algorithm gets told when the answer is wrong, but does not get told how to correct it. It has to explore and try out different possibilities until it works out how to get the answer right. The goal of reinforcement learning is to learn how to act or behave in a given situation for the given reward or penalty signals. In this type of learning, a state for current status is defined, and an environment, usually a criterion function, evaluates the current state to generate a proper reward or penalty action through a set of policies. Instead of having exact target values, it learns with critics. Therefore, reinforcement learning is sometimes called learning with a critic because of this monitor that scores the answer, but does not suggest any improvements [52].

A robot navigating in an environment in search of a goal location is one possible application area of reinforcement learning. At any time, the robot can move in one of a number of directions. After a number of trial runs, it should learn the correct sequence of actions to reach the goal state from an initial state, doing this as quickly as possible and without hitting any of the obstacles.

One factor that makes reinforcement learning harder is when the system has unreliable and partial sensory information. For example, a robot equipped with a video camera has incomplete information, and thus, at any time, is in a partially observable state and should decide taking into account this uncertainty. For example, it may not know its exact location in a room, but only that there is a wall to its left. A task may also require a concurrent operation of multiple robots that should interact and cooperate to accomplish a common goal [1].

3.2 Artificial Neural Network

An Artificial Neural Network is a computational model inspired by networks of biological neurons. In animals, learning occurs within the brain. While the brain is an impressively powerful and complicated system, the basic building blocks that it is made up of are fairly simple and easy to understand. In computational terms, the brain deals with noisy and even inconsistent data, and produces very quick answers that are usually correct even from very high dimensional data, such as images.

Neuron

A neuron is an electrically excitable cell that communicates with other cells via specialized connections called synapses. There are hundreds of billions of neurons in a human brain [7]. The input to the neuron is provided by dendrites, a number of ramifying branches, which continually monitor changes in the external and internal environment. The output of the neuron is provided by a long fiber called the axon. The general operation of a neuron is that transmitter chemicals within the fluid of the brain raise or lower the electrical potential inside the body of the neuron. If this membrane potential reaches some threshold, the neuron spikes (or fires), and a pulse of fixed strength and duration is sent down the axon. The axons divide into connections to many other neurons, connecting to each of these neurons in a synapse. Each neuron is typically connected to thousands of other neurons [7]. A picture of two neurons can be seen in Figure 3.2.

Each neuron can be viewed as a separate processor, performing a very simple computation, which is deciding whether or not to fire. This makes the brain a massively parallel computer. The basic principle of learning is to modify the strength of synaptic connections between neurons, and to create new connections.

Changes in the strength of synaptic connections are proportional to the correlation in the firing of the two connecting neurons. So if two neurons consistently fire simultaneously, then any connection between them will change in strength, becoming stronger. However, if the two neurons never fire simultaneously, the connection between them will die away. The idea is that if two neurons both respond to something, then they should be connected [32].

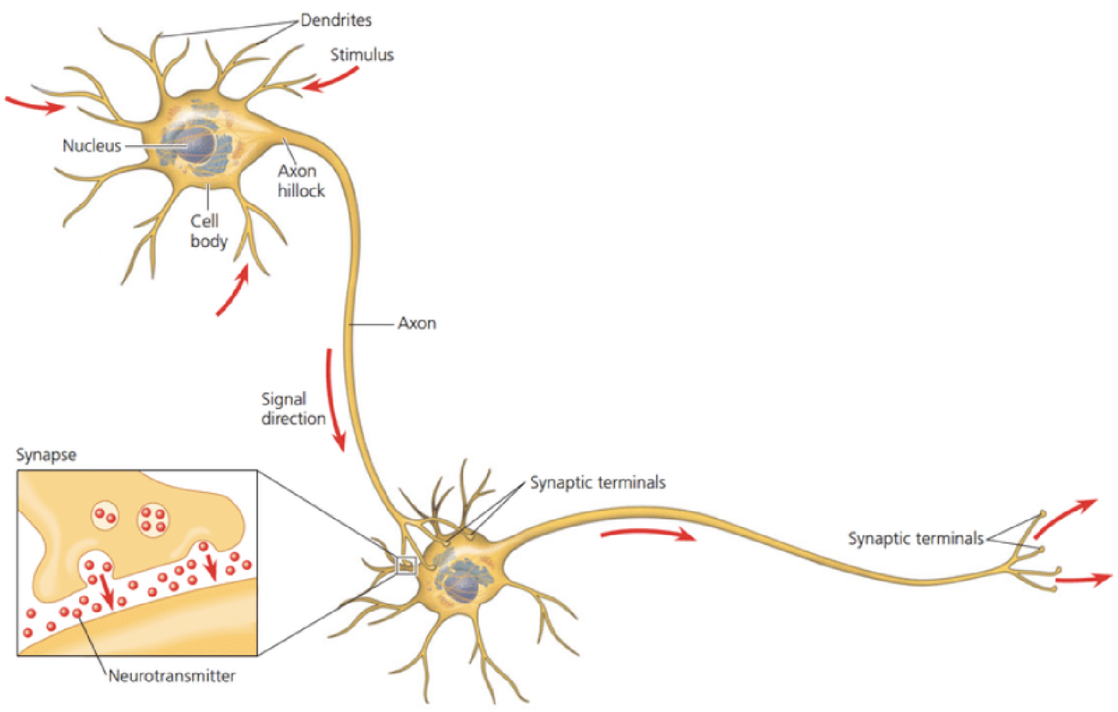


Figure 3.2: A neuron consisting of a cell body, an axon and multiple dendrites creating a connection to another neuron [33].

Perceptron

Perceptrons were invented as simple computational models of neurons. A perceptron is a neural network with one artificial neuron. It takes many inputs and has one output. Its first half consists of a vector of weights $w = [w_1 \dots w_m]$, one for each input, plus distinguished weight, b, called the bias. Weights represent weighted connections between neurons. These weights are equivalent to the synapses in the brain. Weights and bias are called the parameters of the perceptron. The basic operation performed by the perceptron is to multiply the values of each input x_i by its weight w_i, sum the results up and add the bias. It can be written as:

$$z = b + \sum_{i=1}^{n} x_i w_i \tag{3.1}$$

where $x = [x_1 \dots x_n]$ is the input vector. The bias is added for cases where all of the inputs are zero. In such a case, it does not matter what the weights are, since zero times anything equals zero. The only way to control the output of the perceptron is through the bias. It represents an extra input weight to the perceptron, with the value of input always being fixed.

The second half of the work of the perceptron is to decide whether to produce output of 1 or output of 0 depending on whether the value z is above some threshold θ. This is also known as an activation function:

$$a = \sigma(z) = \begin{cases} 1 & \text{if } z > \theta \\ 0 & \text{otherwise} \end{cases} \tag{3.2}$$

Perceptrons are binary classifiers, so 1 indicates that x is a member of the class, and 0 not a member [13]. A graphical representation of the perceptron is shown in Figure 3.3.

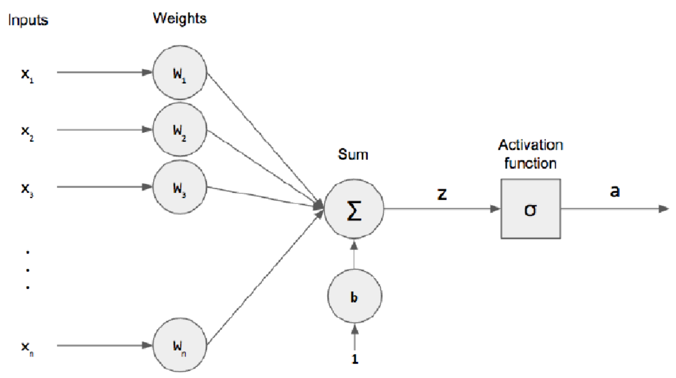
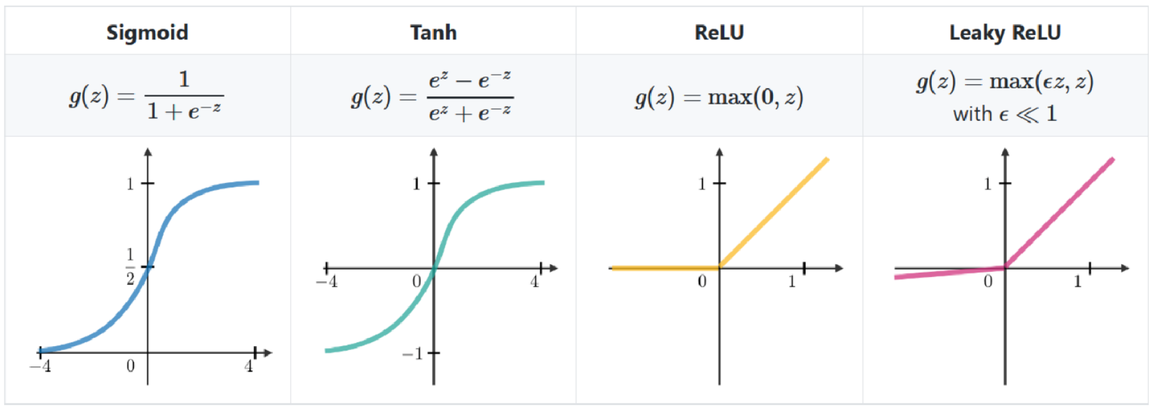


Figure 3.3: A perceptron with n inputs. A weighted sum z of the inputs and the bias is passed through an activation function σ that gives an output of 1 if the sum is greater than the defined threshold and an output of 0 otherwise [16].

The activation function σ is to be selected on the basis of the nature of the problem. It mathematically defines the properties of perceptrons. It can be any step function or non-linear sigmoid function, depending on the problem. The most common activation functions are shown in Figure 3.4 below.



Sigmoid	Tanh	ReLU	Leaky ReLU
$g(z) = \frac{1}{1+e^{-z}}$	$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$g(z) = \max(0, z)$	$g(z) = \max(\epsilon z, z)$ with $\epsilon \ll 1$

Figure 3.4: Different types of functions used as activation functions of perceptrons [5][6][4][2].

The perceptron needs to be trained before it can be used. The training algorithm works by iterating over the training data several times, adjusting the weights to increase the number of correctly identified examples. Each pass through the data is called an epoch. The corresponding input of the perceptron is set, and then Equations 3.1 and 3.2 are used to calculate the output, which is then compared to the target that is known to be the correct answer for this input. Loss or distance functions are defined between the current output vector and the target vector for each input vector, and optimization is performed to minimize the loss over all training examples.

If the answer of the perceptron is correct, there are no adjustments, but if the answer is incorrect, the perceptron needs to have its weights changed. Some of the weights will be too big if the perceptron produced 1 when it should not have, and too small if it did not produce 1 when it should. Therefore, the difference between the target t, which is the anticipated answer, and the output y of the perceptron is computed. If the result is positive, then the perceptron should have produced 1 and it did not, so the weights are made bigger, and vice versa if it is negative. The rule for updating a weight w_i is:

$$w_i \leftarrow w_i + \eta(t - y) \cdot x_i \tag{3.3}$$

where η is a parameter called the learning rate. The value of the learning rate decides how much the weight should change by, and thus how fast the network learns. If the learning rate is missed out, the weights change a lot whether there is a wrong answer, which tends to make the network unstable, so that it never settles down. The cost of having a small learning rate is that the weights need to see the inputs more often before they change significantly. However, it will be more stable and resistant to noise and inaccuracies in the data [32]. An element of the input could be negative, which would switch the values over, therefore, the difference in Equation 3.3 is multiplied by x_i, which makes the value of the weight negative as well.

Perceptrons are linear models. They try to separate out the cases where they should produce an output of 1 from those where they should not. This is done by finding a straight generalization line in 2D, a plane in 3D, or a hyperplane in higher dimensions. This line is called the decision boundary or discriminant function [32]. An example of one is given in Figure 3.5. The cases where there is a straight line are called linearly separable cases.

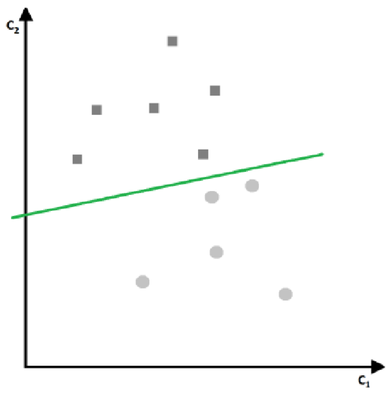


Figure 3.5: A decision boundary separating two classes of data.

Multi-Layer Perceptrons

Linear models are easy to understand and use. They can identify straight lines, planes or hyperplanes, but the majority of problems are not linearly separable. Learning in the neural network happens in the weights, and thus, adding more neurons between the input nodes and the outputs will make more complex neural networks, such as the one shown in Figure 3.6. Adding extra layers of nodes makes a neural network more powerful. All these nodes are interconnected, so the output of one node is connected to the inputs of all nodes in the next layer.

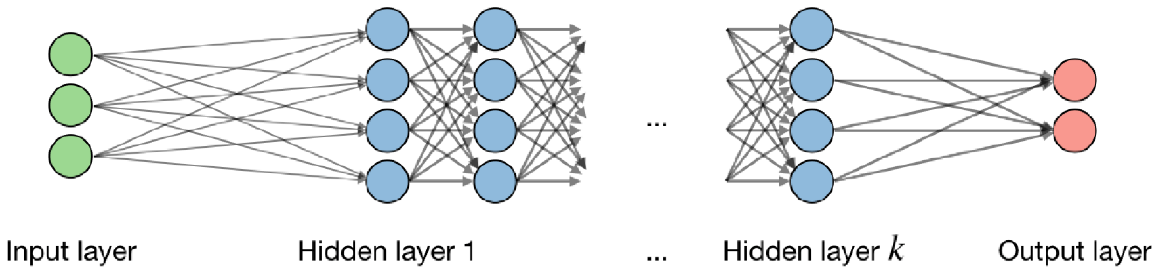


Figure 3.6: A neural network consisting of multiple layers of interconnected neurons [3].

To train this network, the difference between the targets and outputs can be computed, but it is not possible to find out which weights were wrong and in which layer. Nor is it possible to determine what the correct activations are for neurons in the middle of the network. This fact gives the neurons in the middle of the network their name. They are called the hidden layer, because it is not possible to examine and correct their values directly.

Learning process uses two popular algorithms named *feed-forward* and *backpropagation*. The term feed-forward describes how the neural network processes and recalls patterns. In a feed-forward neural network, neurons are only connected forward. Each layer of the neural network contains connections to the next layer, but there are no connections back. In this way, values are fed forward. The term backpropagation describes how this type of neural network is trained. Backpropagation is a form of supervised training. It calculates the error by comparing the anticipated outputs against the actual outputs for a given input, and propagates them back to the earlier layers. The weights of the various layers are adjusted backwards from the output layer to the input layer to reduce the value of error. It is a form of gradient descent (Figure 3.7). If a function is differentiated, we get the gradient of that function, which is the direction along which it increases and decreases the most. So if we differentiate an error function, we get the gradient of the error. Following the function in the direction of the negative gradient will minimise the error, and that is the purpose of learning [40].

There is no theory for choosing the number of hidden nodes or the number of hidden layers. The only way is to experiment by training networks with different numbers of hidden nodes, and then choosing the one that gives the best results. The backpropagation algorithm can be used for a network with as many layers as needed, although with an increasing number of layers it gets progressively harder to keep track of which weights are being updated at any given time.

For a network with one hidden layer, there are $(m + 1) \cdot n + (n + 1) \cdot p$ weights, where m, n, o are the number of nodes in the input, hidden and output layers, respectively. Bias nodes also have adjustable weights, so they must be taken into account (the extra +1s). This is a potentially huge number of adjustable parameters that are needed to be set during the training phase. The more training data there is, the better the learning, although the time required for learning increases.

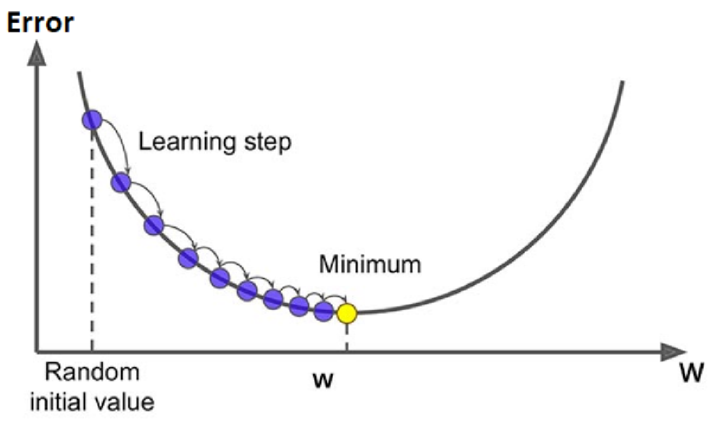


Figure 3.7: Gradient descent is an optimization algorithm for finding the minimum of a function. To find the local minimum of a function, proportional steps to the negative of the gradient are taken at the current point [3].

Overfitting and Underfitting

The main purpose of using a neural network is to perform well on new, previously unseen inputs. This ability of the neural network is called generalization. Therefore, we want the generalization error, to be as low as possible, where the generalization error is defined as the expected value of the error on a new input [19].

The neural network has to be sufficiently trained to generalize well. However, there is at least as much danger in over-training the network as there is in under-training it. If the network is trained for too long, then it will overfit the data, which means that the network has learned about noise and inaccuracies in the data as well as the actual function. The network will be too complicated, and it will not be able to generalize. This is shown in Figure 3.8.

A model can be controlled whether it is more likely to overfit or underfit by altering its capacity. Capacity is the number of learnable parameters. Machine learning algorithms will generally perform best when their capacity is appropriate for the true complexity of the task they need to perform and the amount of training data they are provided with. Models with insufficient capacity are unable to solve complex tasks. Models with high capacity can solve complex tasks, but when their capacity is higher than needed to solve the present task, they may overfit [19]. The relationship between capacity and generalization error is shown in Figure 3.9.

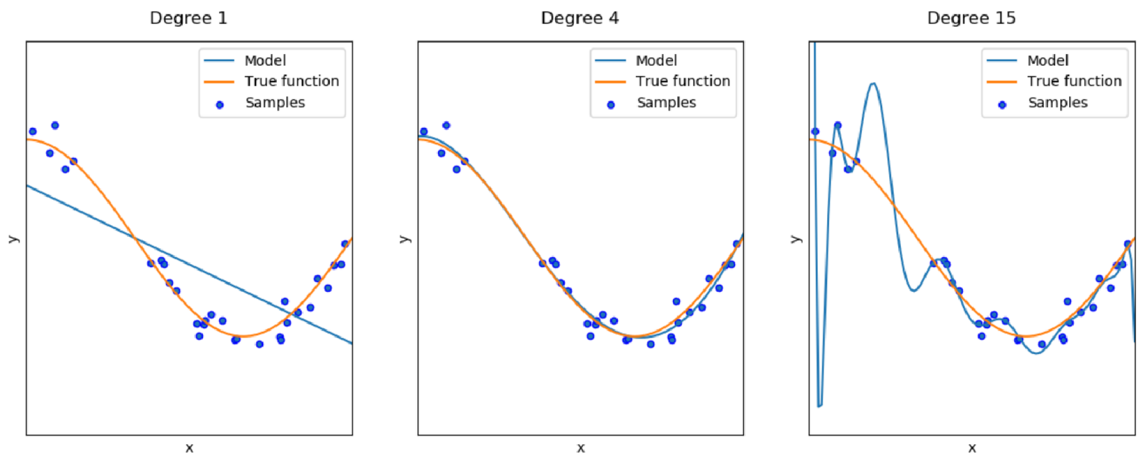


Figure 3.8: The graph shows a part of the cosine function that should be approximated. In addition, samples from the true function and its approximations are displayed. The models can use the polynomial functions of different degrees. A linear function on the left is not sufficient to fit the training samples. This is called underfitting. A polynomial of degree 4 in the center approximates the cosine function almost perfectly, but for higher degrees on the right, the model will overfit the training data, since the solution passes through all the training samples exactly [45].

There are several methods that prevent a model from overfitting. They are called regularization, and they are based on constraining the amount of information that the model is allowed to store. If a network can only afford to memorize a small number of patterns, it will force the network to focus on the most prominent patterns, which have a better chance of generalizing well. These methods are described below:

- **Reducing the size of the network** is the simplest way to prevent overfitting. To reduce the size of the network, the number of its learnable parameters is decreased. A network with more parameters has more memorization capacity, and therefore, can easily learn a perfect mapping between training samples and their targets without any generalization power.
- **Adding weight regularization**. A model where the distribution of parameter values has less entropy is less likely to overfit than a complex one. Thus a common way to reduce overfitting is to force its weights to take only small values, which makes the distribution of weight values more regular. This is called weight regularization, and it is done by adding a cost associated with having large weights to the loss function of the network. There are two types of weight regularization:
  1. *L1 regularization* – the added cost is proportional to the absolute value of the weight coefficients
  2. *L2 regularization* – the added cost is proportional to the square of the value of the weight coefficients
- **Adding dropout**, which is one of the most effective and most commonly used regularization techniques for neural networks. Dropout, applied to a layer, consists of randomly dropping out a number of output values of the layer during training. The

dropout rate is the fraction of the values that are zeroed out. At test time, no units are dropped out. Instead, the output values of the layer are scaled down by a factor equal to the dropout rate, to balance for the fact that more units are active than at training time [14].

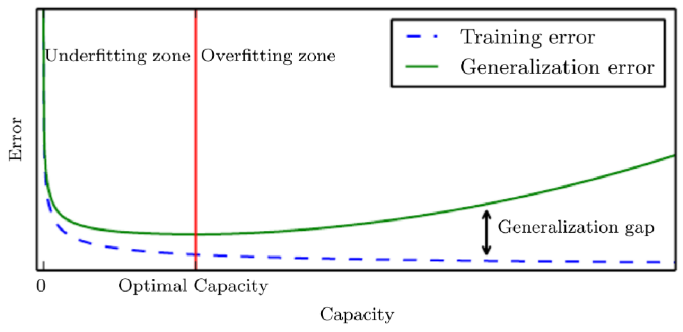


Figure 3.9: The graph shows the relationship between capacity and generalization error. At the left end of the graph, both the training error and the generalization error are high. This is the underfitting zone. As capacity increases, training error decreases, but the gap between training and generalization error also increases. The size of this gap eventually outweighs the decrease in training error, and it gets to the overfitting zone, where capacity is too large, above the optimal capacity [19].

Training, Testing and Validation

We should have at least two and preferably three sets of problem examples. The first is the training set. It is used to adjust the parameters of the model. In order to decide when to stop learning, we have to check how well the network is learning during the training. We can not use the training data for this because we would not be able to detect overfitting. Therefore, we keep the second dataset back, called the validation set. This set is used to validate the learning so far. This is known as cross-validation in statistics [32]. Whenever an artificial neural network is trained, it should be tested how well it works, but it is not sensible to test it using the same data on which it was trained because, it would not tell us anything at all about how well the network generalises nor anything about whether or not overfitting had occurred. Therefore, we must keep the third set, called the test set, which we do not use for training. The only problem is that it reduces the amount of data available for training. The exact proportion of training to testing to validation data is up to us, but it is typical to do something like 60:20:20 [32].

3.3 Deep Learning

Deep learning is a specific subfield of machine learning. It puts an emphasis on learning successive layers of increasingly meaningful representations. The word deep in deep learning is not a reference to any kind of deeper understanding achieved by this approach, but rather, it stands for this idea of successive layers of representations. How many layers contribute to a model of the data is called the depth of the model [14].

Convolutional Neural Network

Convolutional neural networks, or CNNs, are a specialized kind of neural network for processing data that has a known grid-like topology. The core element of convolutional neural networks is data processing using a mathematical operation called convolution. Convolution of any signal with another signal produces a third signal that may reveal more information about the signal than the original signal itself. For example, by convolving a grayscale image as a 2D signal with another signal, generally called a filter or kernel, an output signal can be obtained that contains edges of the original image, which may be useful for several applications.

The most general form of convolution is an operation on two functions f and g of a real-valued argument. The convolution operation is typically denoted with an asterisk [41]:

$$v(t) = (f * g)(t) \tag{3.4}$$

In machine learning applications, the input is usually a multidimensional array of data, and the kernel is usually a multidimensional array of parameters that are adapted by the learning algorithm. We can use the following Equation 3.5 to get the value V of the convolution of an image I at a position x, y, and a kernel K [41]:

$$V(x, y) = (I * K)(x, y) = \sum_m \sum_n I(x + m, y + n)K(m, n) \tag{3.5}$$

An example of such a convolution is shown in Figure 3.10.

The main difference between a fully connected layer, found in a typical neural network, and a convolution layer is that fully connected layers learn global patterns involving all pixels, whereas convolution layers learn local patterns, in the case of images, patterns found in small 2D windows of the input. The patterns CNNs learn are translation invariant. For example, after learning a certain pattern in the middle of a picture, a convolutional neural network can recognize it anywhere. A fully connected network would have to learn a new pattern if the existing one appeared at a new location. This makes convolutional networks data-efficient because they need fewer training samples to learn representations that have generalization power.

CNNs can also learn spatial hierarchies of patterns. The first convolution layer will learn small local patterns such as edges, the second convolution layer will learn larger patterns made of the patterns of the first layers, and so on. This allows convolutional networks to efficiently learn increasingly complex and abstract visual concepts to represent the visual world.

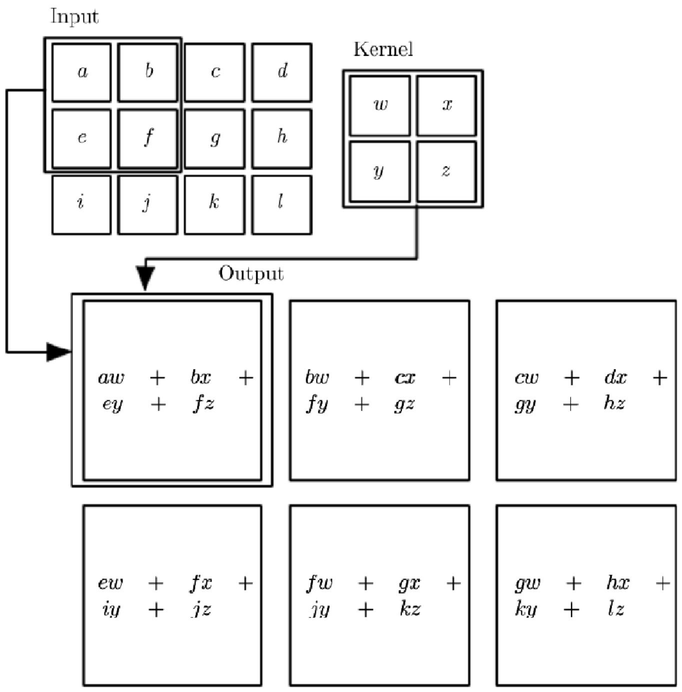


Figure 3.10: An example of 2D convolution. The output is restricted to only positions where the kernel lies entirely within the image [19].

Generative Adversarial Network

Generative adversarial network, or GAN, is an unsupervised deep learning machine, introduced by Ian Goodfellow in 2014 [20]. It enables the generation of fairly realistic synthetic images by forcing the generated images to be statistically almost indistinguishable from real ones. This type of neural network is based on two models:

1. **Generator network** that takes as input a random vector and decodes it into a synthetic image.
2. **Discriminator network** that takes as input a real or synthetic image and predicts whether the image came from a training set or was created by the generator network.

A GAN chains the generator and the discriminator together:

$$GAN(x) = Discriminator(Generator(x)) \tag{3.6}$$

The generator is trained to be able to fool the discriminator to the extent that it is impossible for the discriminator to distinguish between real and fake images. It evolves toward generating increasingly realistic images as training goes on. Meanwhile, the discriminator is constantly adapting to the gradually improving capabilities of the generator. Therefore, both models are being trained to best the other. Once training is over, the generator is capable of turning any point in its input space into a believable image. Figure 3.11 shows a diagram of a generative adversarial network.

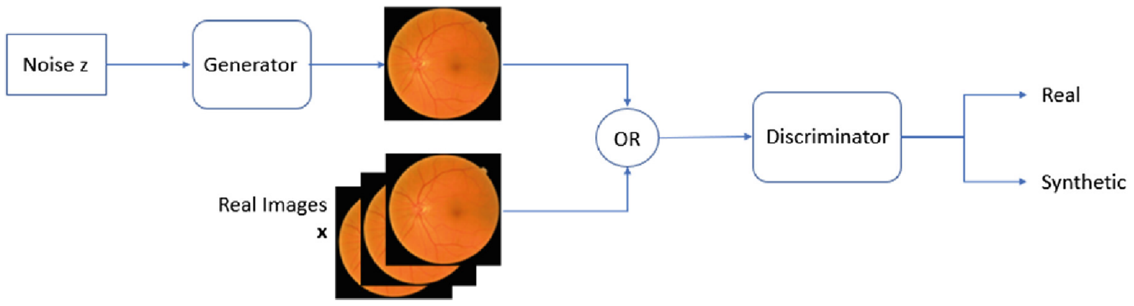


Figure 3.11: The generator transforms random vectors (noise) into images and the discriminator tries to distinguish between real and synthetic images [50].

A GAN is a system where the optimization minimum is not fixed. Normally, gradient descent consists of rolling down hills in a static loss landscape. But with a GAN, every step taken down the hill changes the entire landscape a little. It is a dynamic system where the optimization process is seeking not a minimum, but an equilibrium between the two models. For this reason, GANs are notoriously difficult to train. Getting a GAN to work requires lots of careful tuning of the model architecture and training parameters [14].

Chapter 4

Proposed Solution

Generating realistic-looking images of the retina is not an easy task. The retina contains many structures that have a certain shape and color, and there are also dependencies between their locations. For this reason, I chose the image-to-image translation approach, where the generator is provided with a black and white image of the bloodstream, from which a synthetic image of the retina is subsequently generated. In this way, the generator is prevented from generating an unrealistic bloodstream where, for example, some vessels are not connected to each other, are too wide or, conversely, there is a minimum of blood vessels in the generated retina. Conditional generative adversarial networks are used to implement image-to-image translations, on which I based my solution. This type of network is described below, followed by a description of the proposed system for generating synthetic images of the retina.

4.1 Conditional Generative Adversarial Network

GANs are generative models that learn mapping from random noise vector z to output image y, G: $z \rightarrow y$. In contrast, conditional GANs learn mapping from observed image x and random noise vector z to y, G: $\{x, z\} \rightarrow y$. The generator G is trained to produce outputs that cannot be distinguished from real images by an adversarially trained discriminator, which is trained to do as well as possible at detecting fake images of the generator [25]. This training procedure is shown in Figure 4.1.

Generator

A defining feature of image-to-image translation problems is that they map a high resolution input grid to a high resolution output grid. In addition, the input and output differ in surface appearance, but both are renderings of the same underlying structure. Therefore, structure in the input is roughly aligned with structure in the output.

The generator uses an encoder-decoder network. In such a network, the input is passed through a series of layers that progressively downsample, until a bottleneck layer, at which point the process is reversed. This network requires that all information flow pass through all the layers, including the bottleneck. For many image translation problems, there is a great deal of low-level information shared between the input and output, and it would be desirable to shuttle this information directly across the network [25].

To give the generator a means to circumvent the bottleneck for information like this, skip connections are added, following the general shape of a U-Net [44]. U-Net is a convolutional neural network that was developed for biomedical image segmentation. Its typical shape, for which it got its name, is shown in Figure 4.2.

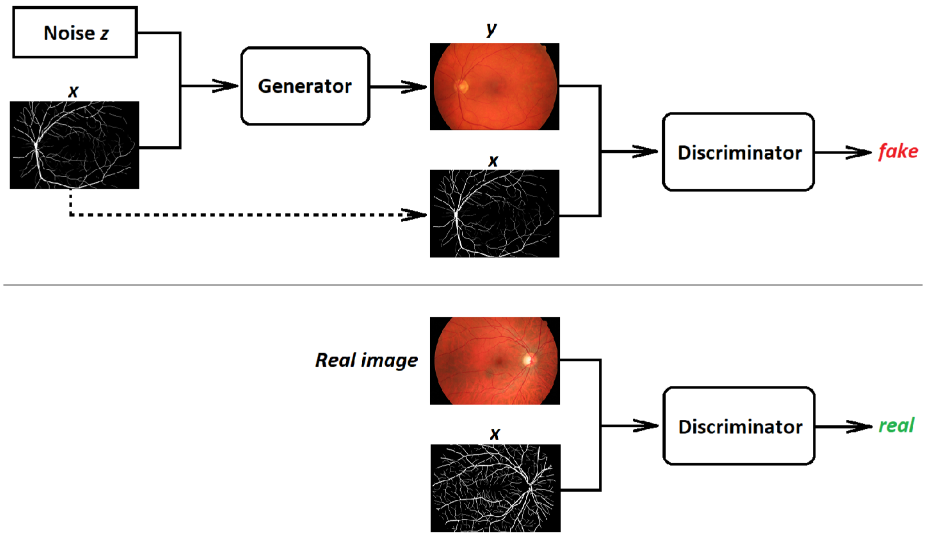


Figure 4.1: Training a conditional GAN to map blood vessels to retinal images. The discriminator learns to classify between fake and real tuples. The generator learns to fool the discriminator. Unlike an unconditional GAN, both the generator and discriminator observe input images of blood vessels.

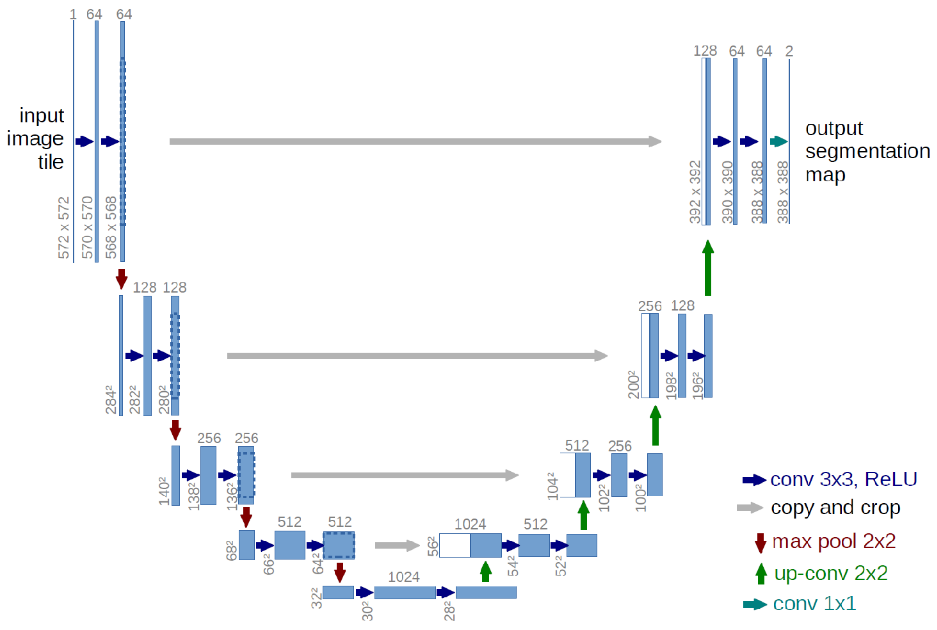


Figure 4.2: U-net architecture. Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The dimensions are provided at the lower left edge of each box. White boxes represent copied feature maps. The arrows denote different operations, where the grey ones represent skip connections [44].

Discriminator

It is well known that the L2 loss and L1 produce blurry results on image generation problems [30]. Although these losses fail to encourage high-frequency crispness, in many cases they nonetheless accurately capture low frequencies. In such cases, L1 enforces correctness at the low frequencies. Therefore, the GAN discriminator is restricted to only model high-frequency structure, relying on an L1 term to force low-frequency correctness. In order to model high frequencies, it is sufficient to examine the structure in local image patches. In this way, the discriminator architecture is designed to only penalize structure at the scale of patches, where it tries to classify if each $N \times N$ patch in an image is real or fake. This discriminator is run convolutionally across the image, averaging all responses to provide the ultimate output of the discriminator.

N can be much smaller than the full size of the image and still produce high quality results. This is advantageous because a smaller network has fewer parameters, runs faster, and can be applied to arbitrarily large images. Such a discriminator effectively models the image as a Markov random field, assuming independence between pixels separated by more than a patch diameter. Therefore, it is called Markovian discriminator or PatchGAN [25].

4.2 Synthetic Retinal Image Generator

Synthetic Retinal Image Generator, or SRIG, is the proposed system for generating synthetic retinal images with a resolution of 1024×1024 pixels. The same width and height of the input and output was chosen because the retina has a round shape. I also tried to create an extended version of this system that would produce images in a higher resolution of 2048×2048 or more, but I encountered a problem where I was running out of memory during training. That is why the final resolution is 1024×1024.

SRIG is a conditional neural network that generates color images of entire retinas from black and white images containing only segmented blood vessels. SRIG, like GANs, consists of two parts: a generator and a discriminator, where these two models are trained simultaneously by an adversarial process. Once the whole system achieves the desired results, the discriminator is no longer needed and only the generator and its learned weights are used to generate new images. To distinguish real images from fake (generated) images, values 1 and 0 are used, where 1 represents the real image and 0 fake. The mapping of the input to the output by the generator is schematically shown in Figure 4.3. Both the generator and the discriminator use binary cross-entropy loss as their loss function, because there are only two classes into which images are classified – real and fake. Therefore, binary cross-entropy is described in more detail here. This section also describes the architecture of the generator and discriminator, which together form the SRIG system.

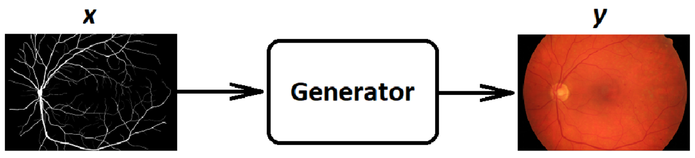


Figure 4.3: The generator learned to map the observed black and white image x to the color output image y, G: $x \rightarrow y$, in order to create a new, previously unseen retina.

Binary Cross-Entropy Loss

Binary cross-entropy loss, also called sigmoid cross-entropy loss, is a sigmoid activation function with a cross-entropy loss. An example of a sigmoid function is the logistic function shown in Figure 4.4 below.

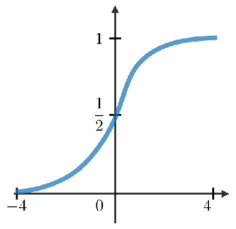


Figure 4.4: The logistic function: $g(z) = \frac{1}{1+e^{-z}}$ with a characteristic sigmoid curve [5].

Cross-entropy loss, or log loss, measures the performance of a classification model, the output of which is a probability value between 0 and 1. Cross-entropy loss increases as the predicted probability deviates from the actual value. Thus, predicting a probability of, for example, 0.01 would result in a high loss value if the actual observed value is 1. A perfect model would have a log loss of 0. An example of cross-entropy loss is given in Figure 4.5. Cross-entropy can be calculated as [47]:

$$-\sum_{c=1}^{M} y_{o,c} \cdot log(p_{o,c}) \tag{4.1}$$

where M is the number of classes of the classification problem, y is a binary indicator, whether the class c is the correct classification for the observation o, and p is the predicted probability that observation o is of class c. Binary cross-entropy loss is used when there are only two classes, in this case whether the image is real or not. Substituting $M = 2$ into Equation 4.1, binary cross-entropy can then be calculated as:

$$-(y \cdot log(p) + (1 - y) \cdot log(1 - p)) \tag{4.2}$$

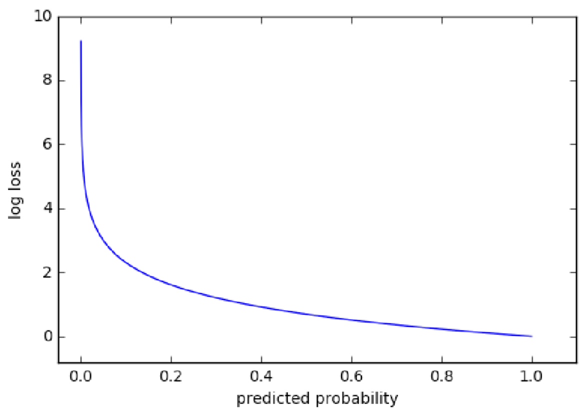


Figure 4.5: The graph above shows a range of possible loss values given a true observation. As the predicted probability approaches 1, log loss slowly decreases. However, as the predicted probability decreases, the log loss increases rapidly [17].

Generator

The architecture of the generator is a modified U-Net. As already mentioned, the generator uses an encoder-decoder network. There are skip connections between the encoder and decoder. This allows us to share information between the input and output, so it better captures the resulting structure that is based on the input structure. Specifically, skip connections are added between each layer i and layer $n - i$, where n is the total number of layers. Each skip connection simply concatenates all channels at layer i with those at layer $n - i$. This architecture can be seen in Figure 4.6, but for simplicity and clarity, the input and output resolution of the image is only 16×16 pixels. The architecture of the actual generator, which generates images in the resolution of 1024×1024 pixels, is presented in Appendix B. More specifically, the encoder is shown in Figure B.1 and the decoder is shown in Figure B.2.

The encoder consists of several blocks, where the total number of these blocks depends on the required output resolution of the generated images. To achieve the bottleneck layer (1×1) at the input resolution of 1024×1024, it is necessary that the number of blocks in the encoder is 10. Each block downsamples the image to half its size, so $log_2(1024)$ blocks are needed for this input resolution, which is 10. Additional blocks may be added to increase the image resolution, but the network will need to be trained again. Each block in the encoder consists of a strided convolution, batch normalization, and leaky ReLU as the activation function of the block. A strided convolution is convolution with a stride. The stride defines the step size of the kernel when traversing the image. Although its default value is usually 1, a stride of 2 is used for downsampling. Batch size represents the number of training samples to work through before the internal parameters of the model are updated. For example, if the batch size is one, the neural network parameters are updated after each sample, and if the batch size is equal to the total number of samples in a dataset, the parameters are not updated until the entire epoch is completed. Batch normalization is a type of layer that can adaptively normalize data even as the mean and variance change over time during training. It works by internally maintaining an exponential moving average of the mean and variance across the batch of the data seen during training. The main effect of batch normalization is that it helps with gradient propagation and thus allows for deeper networks. Instead of ReLU activation, I used leaky ReLU layer because it is similar to ReLU, but it relaxes sparsity constraints by allowing small negative activation values. Side by side comparison between ReLU and leaky ReLU is shown in Figure 3.4 in the previous chapter. A diagram of one encoder block is shown in Figure 4.7.

The number of blocks in the decoder depends on the number of blocks in the encoder, because the encoder and decoder must form a symmetric pattern in order to use the skip connections and generate the same output resolution as the input resolution. Each block in the decoder consists of a transposed convolution, batch normalization and leaky ReLU as the activation function of the block. A transposed convolution, also called deconvolution, represents a transformation going in the opposite direction of a normal convolution, i.e., from something that has the shape of the output of some convolution to something that has the shape of its input, while maintaining a connectivity pattern that is compatible with said convolution. To minimize the risk of overfitting, dropouts were added to ensure that weights are regularized. In this way, each block upsamples the image to twice its size. The last block of the decoder uses *tanh* instead of leaky ReLU as the activation function to get the resulting pixel values. The decoder structure can be seen in Figure 4.8.

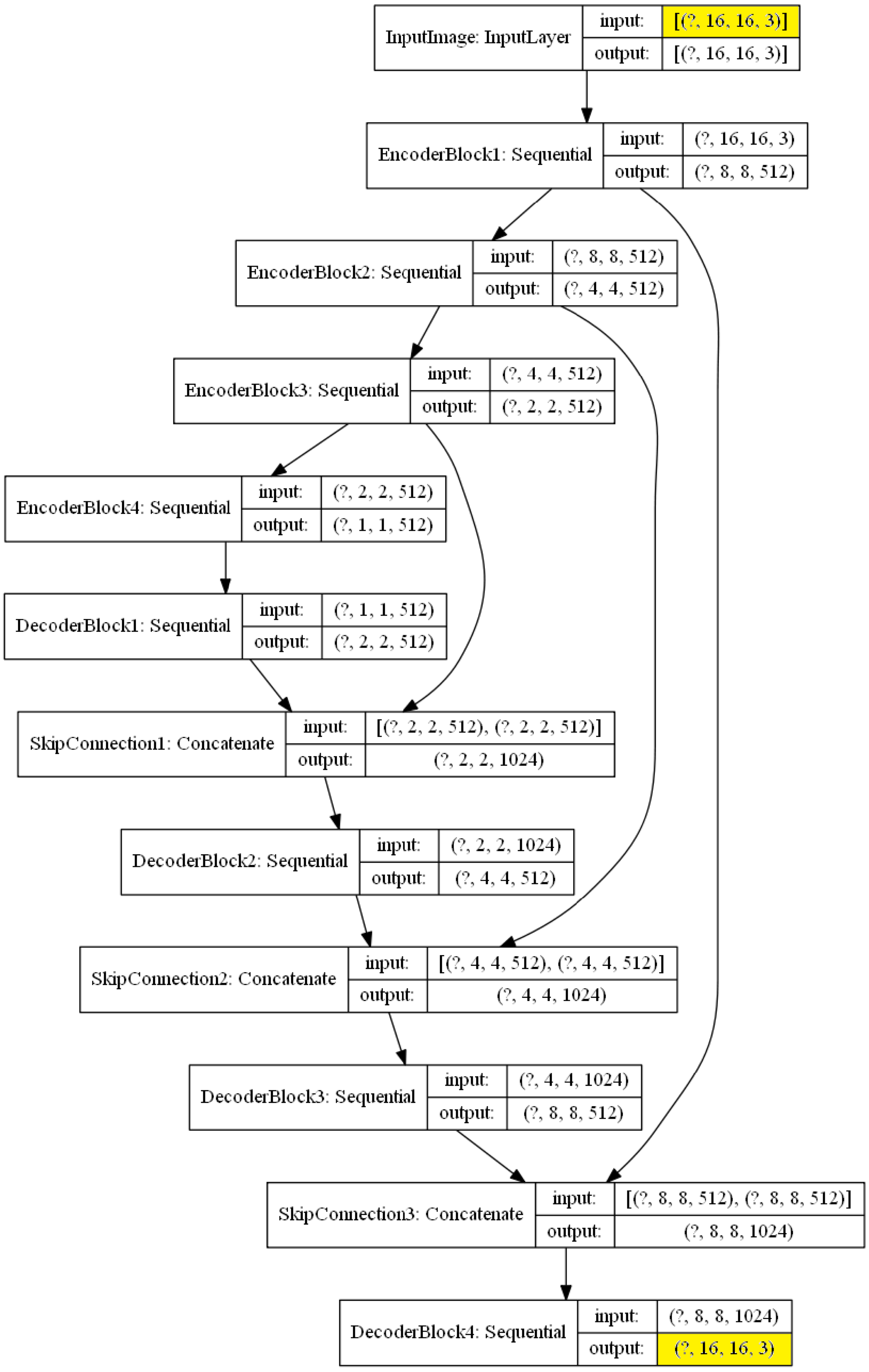

InputImage: InputLayer | input: [(?, 16, 16, 3)] | output: [(?, 16, 16, 3)]
EncoderBlock1: Sequential | input: (?, 16, 16, 3) | output: (?, 8, 8, 512)
EncoderBlock2: Sequential | input: (?, 8, 8, 512) | output: (?, 4, 4, 512)
EncoderBlock3: Sequential | input: (?, 4, 4, 512) | output: (?, 2, 2, 512)
EncoderBlock4: Sequential | input: (?, 2, 2, 512) | output: (?, 1, 1, 512)
DecoderBlock1: Sequential | input: (?, 1, 1, 512) | output: (?, 2, 2, 512)
SkipConnection1: Concatenate | input: [(?, 2, 2, 512), (?, 2, 2, 512)] | output: (?, 2, 2, 1024)
DecoderBlock2: Sequential | input: (?, 2, 2, 1024) | output: (?, 4, 4, 512)
SkipConnection2: Concatenate | input: [(?, 4, 4, 512), (?, 4, 4, 512)] | output: (?, 4, 4, 1024)
DecoderBlock3: Sequential | input: (?, 4, 4, 1024) | output: (?, 8, 8, 512)
SkipConnection3: Concatenate | input: [(?, 8, 8, 512), (?, 8, 8, 512)] | output: (?, 8, 8, 1024)
DecoderBlock4: Sequential | input: (?, 8, 8, 1024) | output: (?, 16, 16, 3)

Figure 4.6: Generator architecture with input and output resolution of 16×16 pixels.

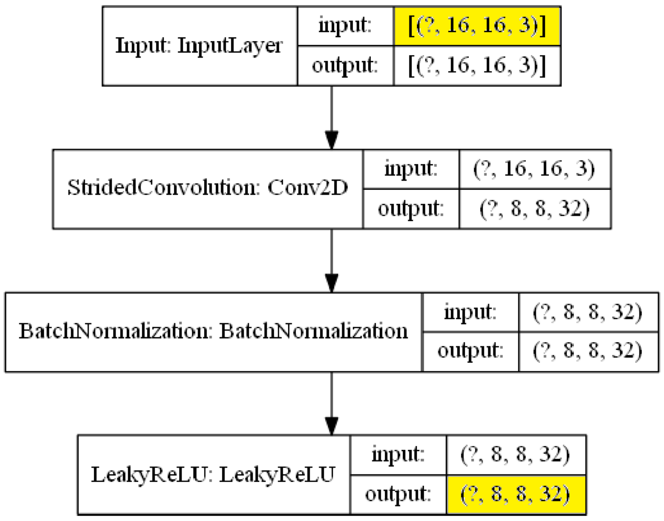


Figure 4.7: Encoder block of the generator, consisting of a convolution layer, batch normalization and leaky ReLU. The data shape is (batch size, height, width, channels), where different channels represent specific colors of the RGB input. After convolution, channels no longer represent colors, but rather stand for filters that encode specific aspects of the input data. The input shape is (?, 16, 16, 3) and the output shape is (?, 8, 8, 32).

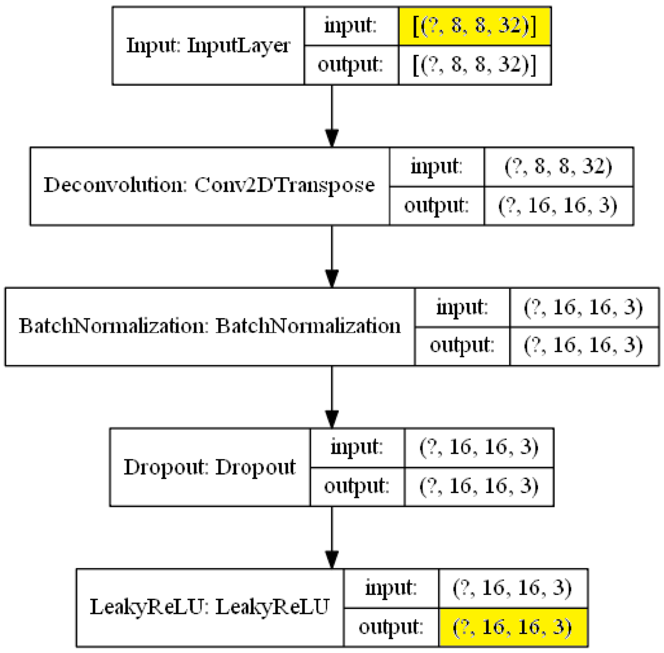


Figure 4.8: Decoder block of the generator, consisting of a transposed convolution layer, batch normalization, dropout and leaky ReLU. The input shape is (?, 8, 8, 32) and the output shape is (?, 16, 16, 3).

Loss of the generator quantifies how well the generator was able to trick the discriminator. If the generator is performing well, the discriminator will classify fake images (or 0) as real (or 1). Therefore, the discriminator decision on the generated images is compared to a set of ones using binary cross-entropy. The previously mentioned L1 loss is added, where L1 loss is the mean absolute error between the generated image and the target image. This allows the generated image to become structurally similar to the target image. The mean absolute error can be calculated as:

$$MAE = \frac{\sum_{i=1}^{n} |g_i - t_i|}{n} \tag{4.3}$$

where g_i is the generated value, t_i is the target value and n is the number of samples. The total loss of the generator thus consists of cross-entropy loss and L1 loss:

$$total\ loss = cross\text{-}entropy\ loss + (L1\ loss \cdot \lambda) \tag{4.4}$$

where the value of λ is 100 [25]. Its purpose is to regulate the impact of L1 loss. A graphical representation of the total loss calculation of the generator is given in Figure 4.9 below.

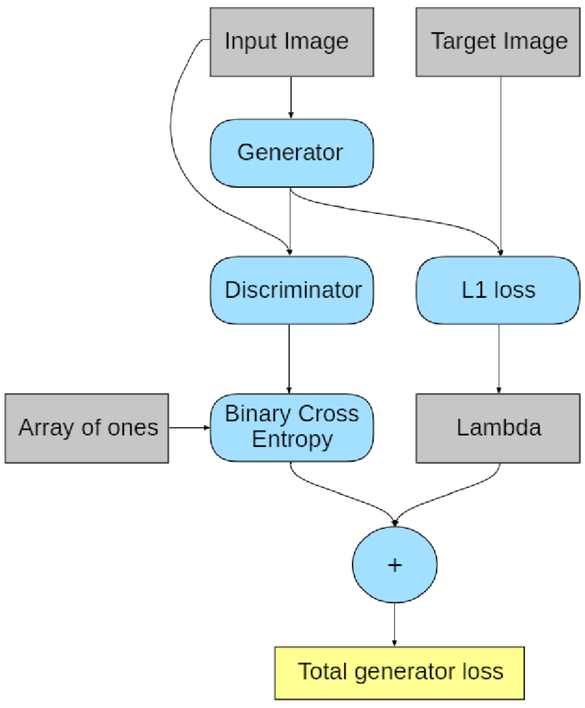
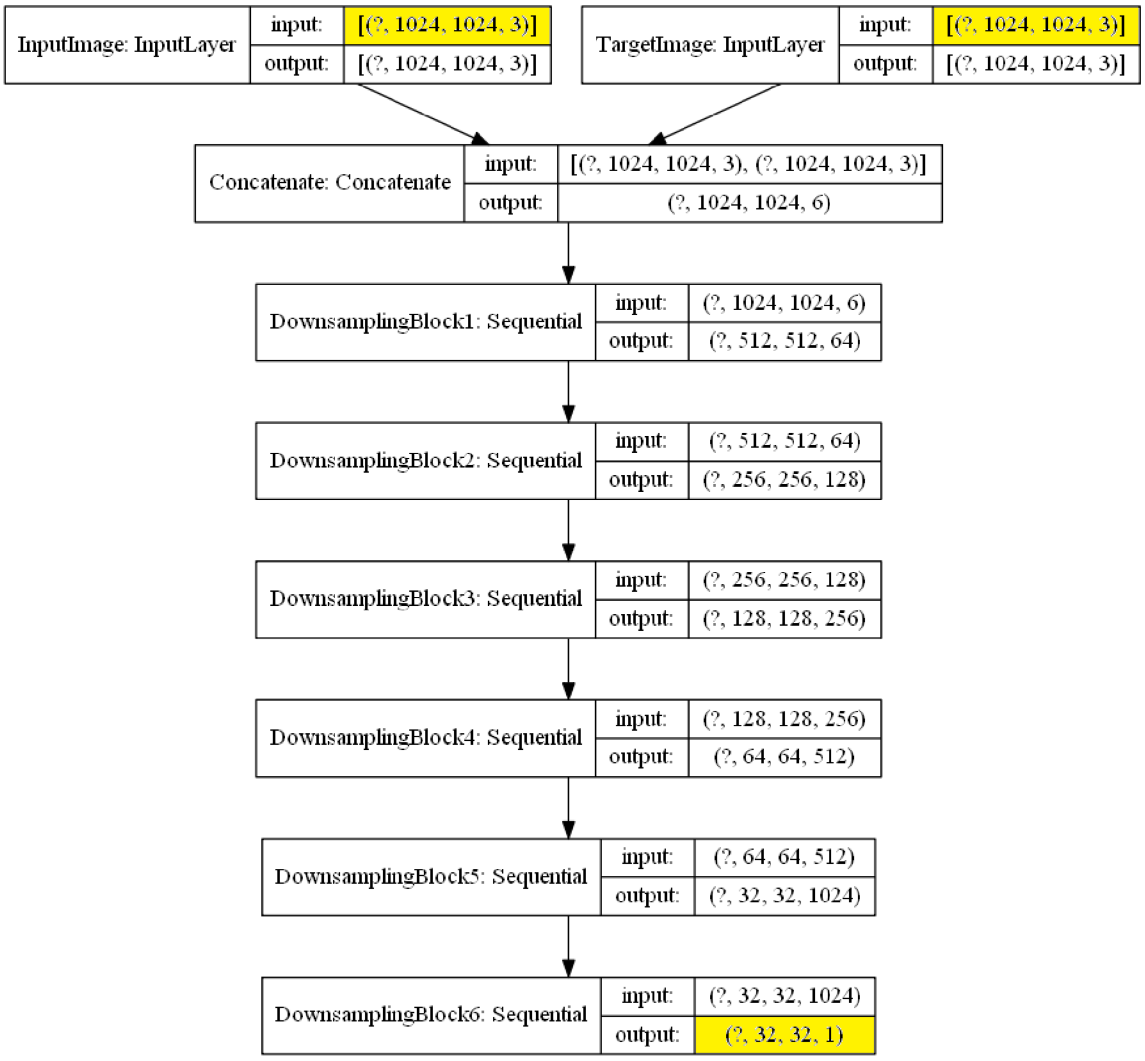


Figure 4.9: The process of calculating the total loss of the generator.

Discriminator

The architecture of the discriminator is a PatchGAN. It is sufficient to examine the structure of an input image in local patches. For this reason the output of the discriminator is a set of patches and not a single response, resulting in higher overall performance. Each patch of the output classifies a given portion of the input image, whether it is real or fake. The input image is therefore downsampled to 32×32 patches, from which the total result is obtained by averaging all patches. The discriminator contains 5 downsampling blocks, because to reduce the input resolution of 1024×1024 to 32×32, the discriminator needs $log_2(\frac{1024}{32})$ blocks, which is 5. Each block consists of a strided convolution that performs the downsampling, followed by batch normalization and leaky ReLU as the activation function, except for the last block, where there is a linear activation function to get the desired output. The architecture of the whole discriminator is shown in Figure 4.10 below. A diagram of one block of the discriminator is shown in Figure 4.11.



InputImage: InputLayer | input: [(?, 1024, 1024, 3)] | output: [(?, 1024, 1024, 3)]
TargetImage: InputLayer | input: [(?, 1024, 1024, 3)] | output: [(?, 1024, 1024, 3)]
Concatenate: Concatenate | input: [(?, 1024, 1024, 3), (?, 1024, 1024, 3)] | output: (?, 1024, 1024, 6)
DownsamplingBlock1: Sequential | input: (?, 1024, 1024, 6) | output: (?, 512, 512, 64)
DownsamplingBlock2: Sequential | input: (?, 512, 512, 64) | output: (?, 256, 256, 128)
DownsamplingBlock3: Sequential | input: (?, 256, 256, 128) | output: (?, 128, 128, 256)
DownsamplingBlock4: Sequential | input: (?, 128, 128, 256) | output: (?, 64, 64, 512)
DownsamplingBlock5: Sequential | input: (?, 64, 64, 512) | output: (?, 32, 32, 1024)
DownsamplingBlock6: Sequential | input: (?, 32, 32, 1024) | output: (?, 32, 32, 1)

Figure 4.10: Discriminator architecture that classifies images with an input resolution of 1024×1024 pixels. It has two images (input, target) on the input, which it concatenates and then progressively downsamples to the output form of 32×32 patches, from which the final prediction is made.

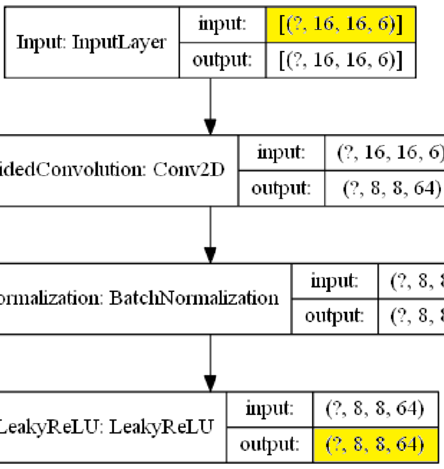


Figure 4.11: One block of the discriminator, consisting of a convolution layer, batch normalization and leaky ReLU. The number of channels on the input represents the channels of the concatenated input and target image.

Loss of the discriminator quantifies how well the discriminator is able to distinguish real images from the fake ones. Predictions of the discriminator on real (target) images are compared to an array of ones, and predictions on fake (generated) images to an array of zeros. The discriminator loss function therefore takes 2 inputs: predictions on generated and target images. The total discriminator loss is:

$$total\ loss = g\_loss + t\_loss \tag{4.5}$$

where g_loss is a binary cross-entropy loss of the generated image and an array of zeros and t_loss is a binary cross-entropy loss of the target image and an array of ones. A graphical representation of the calculation of the total loss of the discriminator is given in Figure 4.12.

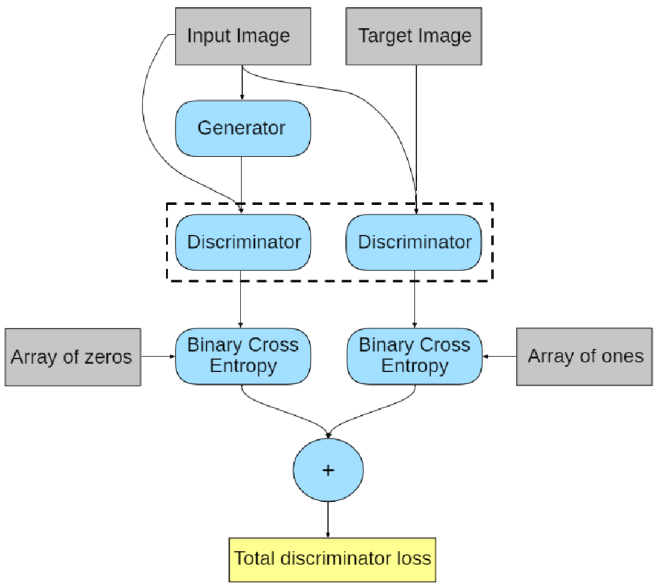


Figure 4.12: The process of calculating the total loss of the discriminator.

Chapter 5

Implementation

This chapter will introduce what technologies are used to implement the proposed system from the previous chapter, followed by a description of the actual implementation of each part.

5.1 Technologies

The proposed program for generating retinal images is implemented in Python programming language using TensorFlow library. I chose Python for several reasons, mainly because it is freely available for academic and commercial purposes, it is multi-platform, and is becoming very popular for both general computing and scientific computing. TensorFlow was chosen because it provides a high-level API that makes it easier to build and train machine learning models without sacrificing speed or performance. With TensorFlow, I could simply focus on the overall logic of the application, instead of dealing with the details of implementing the algorithms.

Python

Python is a scripting language that is strongly typed, but it performs all the declaration and creation of variables for us. Any set of commands or functions in a single source file is known as a module. Python has a fairly small set of commands and is designed to be fairly small and simple to use. Writing extension packages for Python is also simple. It does not require any special programming commands. Any Python module can be imported as a package, as can packages written in C (a low-level programming language), which can significantly increase performance.

Python is not a functional programming language, but it does incorporate some of its concepts alongside other programming paradigms, such as higher order functions. Higher order functions either accept a function as an argument or return a function for further processing. Python has implemented some commonly used higher order functions from functional programming languages that make processing iterable objects like lists and iterators much easier. One of them is the *map()* function, which allows us to apply a function to every element in an iterable object. Another frequently used function is the *filter()* function, which tests each element in an iterable object with a function that returns either *True* or *False* while keeping only those that evaluate to *True*.

TensorFlow

TensorFlow is a free and open-source software library for numerical computation and machine learning using data flow graphs. These structures describe how data moves through a graph or a series of processing nodes. Nodes in the graph represent mathematical operations, and each connection or edge between nodes is a multidimensional data array, or tensor. Nodes and tensors in TensorFlow are Python objects, and TensorFlow applications are themselves Python applications.

The actual math operations, however, are not performed in Python. It uses Python to provide a convenient front-end API for building applications with the framework, but the libraries of transformations that are available through TensorFlow are written as high-performance C++ binaries. The flexible architecture allows us to deploy computation to one or more CPUs or GPUs in a desktop, server, or mobile device with a single API [51].

TensorBoard

TensorBoard is a visualization software that comes with any standard TensorFlow installation. In machine learning, to improve something, we often need to be able to measure it. TensorBoard is a tool for providing the measurements and visualizations needed during the machine learning workflow. This allows us to visualize the model, track metrics, such as loss and accuracy as they change over time, and much more. TensorBoard uses an interactive, web-based dashboard to display this data.

Nvidia CUDA and cudNN

CUDA is a parallel computing platform and programming model developed by Nvidia for general computing on graphical processing units (GPUs). In GPU-accelerated applications, the sequential part of the workload runs on the CPU, while the computationally intensive portion of the application runs on thousands of GPU cores in parallel.

The Nvidia CUDA Deep Neural Network library (cuDNN) is a GPU-accelerated library of primitives for deep neural networks. cuDNN provides highly-tuned implementations for standard routines, such as convolution, normalization, and activation layers. It accelerates widely-used deep learning frameworks, including TensorFlow. This allows TensorFlow to run up to 50% faster on the latest Nvidia GPUs and scale well across multiple GPUs [39].

5.2 Data Preprocessing

Machine learning algorithms tend to learn much more efficiently if preprocessing of inputs and targets is performed before the network is trained. In order to simply read and process data, the *DataLoader* class is implemented. It provides a set of static methods for working with images and creating datasets.

Images that are used as input to a neural network must meet certain requirements because the neural network has a fixed architecture. First, it is resized to match the input resolution of 1024×1024 pixels. Then, because the neural network uses *tanh* as the activation function (Figure 3.4) for the output, the pixel values of the input images need to be between -1 and 1. This also helps to stop the weights from getting too large unnecessarily. Scaling one interval to another is called normalization. The default value range for the pixels is <0,255> when the image is loaded. To scale these values down to

<-1,1>, a simple calculation is performed:

$$normalize(x) = \frac{x}{127,5} - 1 \tag{5.1}$$

where x is the input value of the pixel.

Training Data Preparation

To train the proposed neural network, it is necessary to provide an input image together with the corresponding target image so that the network can learn the mapping between them. It is, therefore, necessary to provide two directories to prepare the input data. One contains images of blood vessels, and the other contains corresponding images of the retinas. Each pair must have the same file name, but may differ in image format. Supported image formats for training data preparation are JPEG, PNG, GIF and TIF. Both specified directories are searched to find all supported images they contain. The *Path* class from the standard library *pathlib* is used to work with paths, which ensures the correct path format on different platforms. From these images, the corresponding pairs (input, target) are then created. In the next step, the individual images of each pair are loaded using the *Image* class from the *PIL* library and are further processed.

Because the architecture of the neural network has a fixed format of input data, it is necessary that the input images also have a fixed format, which requires the dimensions (width and height) of the input images to be the same. For this reason, images are resized to meet these requirements. To prevent information from being lost due to cropping, the image is resized by extending its smaller dimension. The original image is placed in the middle, and the newly created space is filled with black pixels. An example of such an image is shown in Figure 5.1.

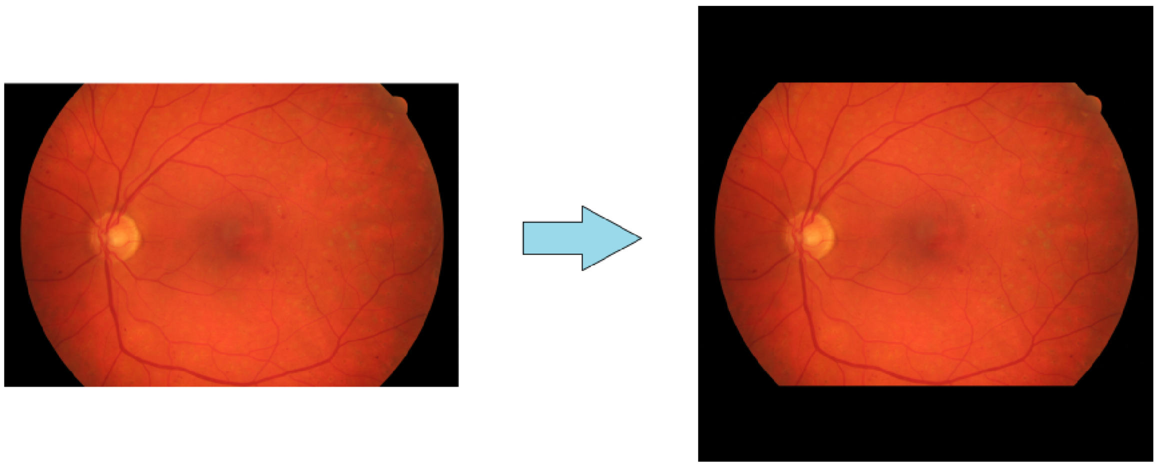


Figure 5.1: The process of resizing an image to the same width and height.

The pairs of images prepared in this way are placed side by side and saved in one file in JPEG format. Figure 5.2 shows an image that is ready to be used as an input image of the neural network for training. The main reason why data is prepared and stored in this way is that during training itself, TensorFlow functions are used to load such images. These functions are better optimized and, therefore, perform better, but do not support as many image formats as the functions in the *PIL* library.

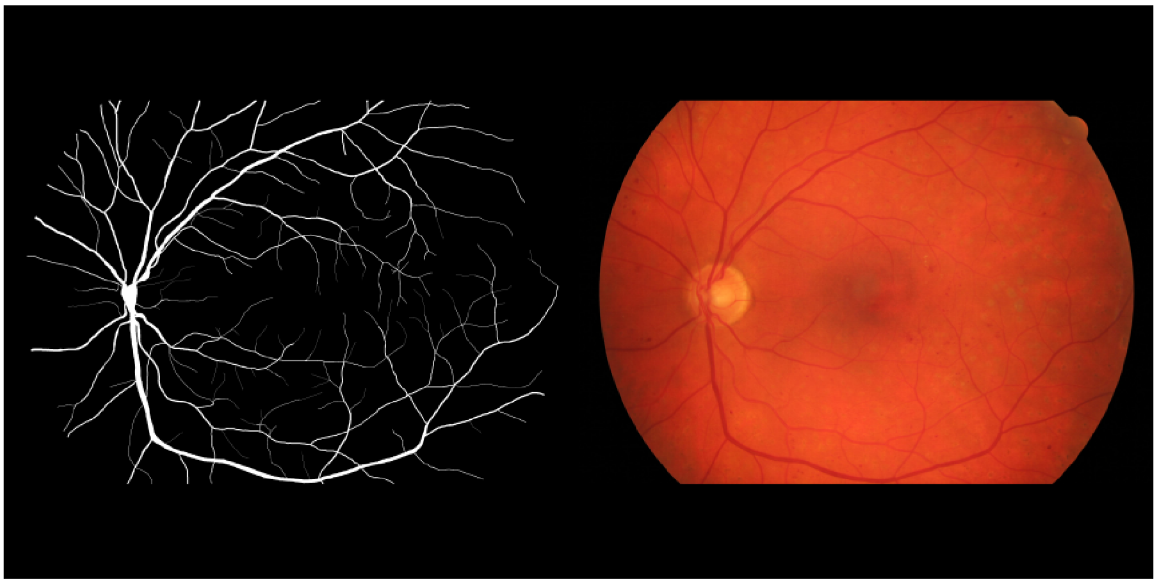


Figure 5.2: An image prepared for training.

Creating a Dataset

A dataset represents a potentially large set of elements. To create one, the TensorFlow class *Dataset* is used, which can apply dataset transformations to preprocess the data. To load the prepared input images, the specified directory is first checked and searched, from which a list of files to be loaded is created. This list is passed to the *Dataset* to initialize it. The *map* function mentioned in Section 5.1 is then used on this dataset to load and preprocess the individual files.

To prevent the neural network from learning the order of the individual images when using the same training set, the data must be well-shuffled before each training. Once all the images have been loaded and preprocessed, the entire dataset is randomly shuffled using the built-in function *shuffle()*. This function also allows the shuffle order to differ for each epoch. The consecutive elements in the dataset are then combined into batches of variable size. The batch size is a parameter that controls the number of training samples to work through before the internal parameters of the model are updated.

To enhance the overall performance, the *cache()* function is used. It caches the elements in the dataset. The first time the dataset is iterated over, its elements are cached either in the specified file or in memory. Subsequent iterations then use the cached data.

Data Augmentation

Data augmentation is the process of generating more training data from existing training samples by augmenting the samples via a number of random transformations that yield believable-looking images. The goal is that at training time, the model will never see the exact same image twice. This helps to expose the model to more aspects of the data and generalize better.

To augment the training data, the images were zoomed in and some of them were flipped horizontally and/or vertically. To zoom inside an image, the image is enlarged and then randomly cropped to its original size. The flipping is relevant because retinal images are not horizontally or vertically symmetrical. An example of augmenting an image is shown in Figure 5.3.

The augmented images are still heavily intercorrelated because they come from a small number of original images. Data augmentation cannot produce new information; it can only modify existing information.

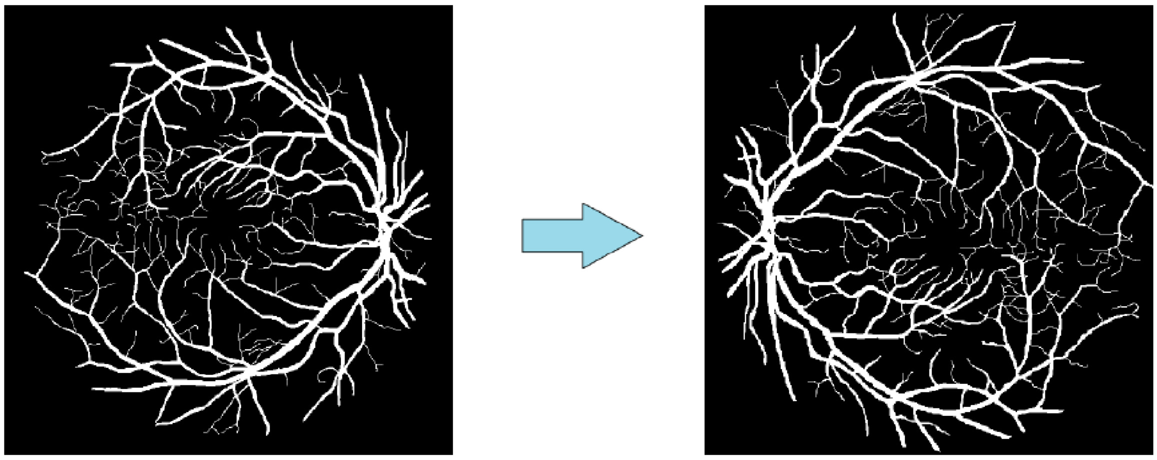


Figure 5.3: Image augmentation. The image on the left is the original image. The image on the right was created by randomly cropping and flipping (both horizontally and vertically) the original image.

5.3 Conditional Generative Adversarial Network

To implement the proposed conditional generative adversarial network from Chapter 4, several classes were created, namely, *Model*, *Generator*, *Discriminator* and *ConGAN*. Their purpose and detailed description is given in the following sections. The corresponding class diagram is shown in Figure 5.4.

Model

Because the generator and the discriminator are models that have certain things in common, the abstract *Model* class was created, which forms the common basis of these models. This class provides a basic interface for working with these models by implementing an object-oriented mechanism – class inheritance. It allows access to individual properties of the model, such as its loss function, optimizer, training variables or the model itself. The *Model* class has two abstract methods that must be implemented by its sub-classes. These are methods for creating the model and calculating the losses.

It also provides an implemented method for updating model weights that is common to both the generator and the discriminator. This method takes the calculated gradients as an input argument and applies them to its trainable variables using the optimizer. Trainable variables represent model weights that are not fixed. An optimizer is a class in TensorFlow that ensures applying the gradients to the variables.

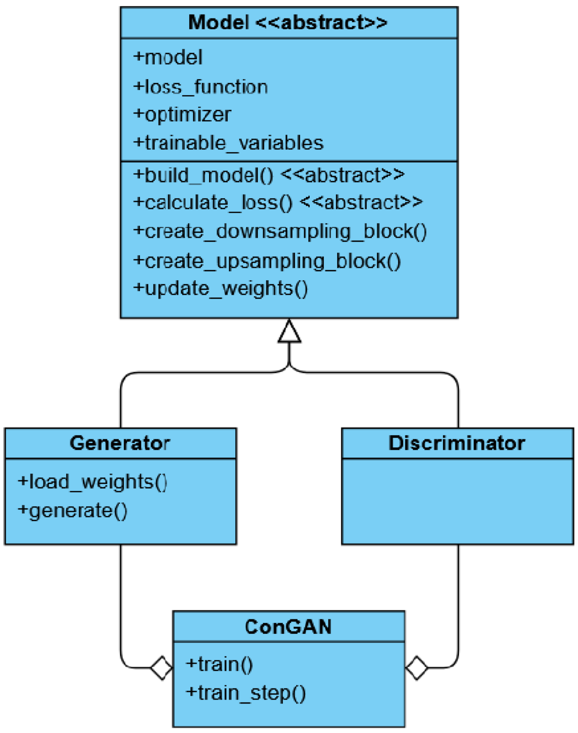


Figure 5.4: Class diagram of the proposed conditional generative adversarial network.

Both the generator and the discriminator use the *Adam* optimizer from TensorFlow. Adam optimization is a stochastic gradient descent method that is based on adaptive estimation of first-order and second-order moments. This method is computationally efficient, has little memory requirement, and is well-suited for problems that are large in terms of parameters [27]. The input parameter of Adam is the learning rate.

Since both the generator and the discriminator are made up of similar blocks, the *Model* class provides two static methods for creating them. One method is used to create a downsampling block, and the other method is used to create an upsampling block.

A downsampling block consists of a convolution layer, batch normalization and leaky ReLU. The *Conv2D* class is used for the realization of the convolution layer. Its main input parameter is the number of filters used for convolution. Another parameter is the size of these filters, and the stride, which is set to 2 for downsampling. The *BatchNormalization* and *LeakyReLU* classes are used to implement batch normalization and leaky ReLU.

An upsampling block consists of a transposed convolution layer, batch normalization, dropout and leaky ReLU. The *Conv2DTranspose* class is used to realize the transposed convolution. Its parameters are the same as in the case of the convolution layer. The stride is also set to 2, but now it is for upsampling. The dropout layer is implemented using the *Dropout* class. This layer randomly sets input units to 0 with the specified frequency at each step during training time. This frequency is an input argument whose values are between 0 and 1. Batch normalization and leaky ReLU are implemented in the same way as in the downsampling block. All of the above-mentioned classes are implemented within TensorFlow. The individual parameters and their values are described in more detail in Section 6.2 in the following chapter.

Generator

The *Generator* class is inherited from the *Model* class. It represents the generative part of the system. As mentioned in the previous section, the generator must implement both abstract methods of the *Model* class.

The first one is a method for calculating the loss of the generator. This method has three input arguments. The first one is the prediction of the discriminator on the generated image, the second is the generated image itself, and the last argument is the target image. To calculate the binary cross-entropy loss, the *BinaryCrossentropy()* function from TensorFlow is used. The discriminator prediction is passed to this function, together with an array of ones created from this prediction using the *ones_like()* function from TensorFlow. To calculate L1 loss, which is the mean absolute error, the TensorFlow functions *reduce_mean()* and *abs()* are used, to which the difference between the pixel values of the target image and the generated image is passed. The value thus obtained is multiplied by the lambda. For the total loss of the generator, these two calculated values are then summed.

The second abstract method that needs to be implemented is the method for creating the model itself. The prepared methods for downsampling and upsampling are used to create individual encoder and decoder blocks. These blocks are stored in two arrays. The array with encoder blocks is iterated over to create the encoder part. In each iteration, the output of one block is connected to the input of another block, and, at the same time, these blocks are stored in an auxiliary array, which represents the skip connections. After creating the encoder part, this auxiliary array is reversed. In the next step, it is iterated over this array and the array with decoder blocks. In each iteration, a skip connection between the encoder and decoder is created using the TensorFlow *Concatenate* layer. This creates the final structure of the generator.

For the purpose of generating synthetic retinal images, the *Generator* class provides two other methods. One method to generate an image and the other to initialize the generator. The first method has three input arguments, the first of which is the input black and white image with segmented blood vessels. The second argument specifies the name of the output file, in which the generated image will be saved. To plot and save images, the *Pyplot* library is used, which, however, requires that the pixel values of the image lie in the interval <0,1>. And because the values of the generated image lie in the interval <-1,1>, it is necessary to normalize them. To do so, the following equation can be used:

$$normalize(x) = \frac{x+1}{2} \tag{5.2}$$

The last optional argument, whose default value is *False*, is a flag indicating whether the generated image should be displayed on the screen or not.

To generate an image that is not just random noise, the generator needs to be initialized. To initialize the generator, the second of the mentioned methods is used. It initializes the generator weights from a checkpoint located in the directory passed as the input parameter of the method. The *Checkpoint* class from TensorFlow is used to work with checkpoints.

Discriminator

The *Discriminator* class is the second class that is inherited from the *Model* class. The discriminator is only used in the training phase, so it provides implementations of only the abstract methods of the parent class that are sufficient to achieve the desired result.

The first is again the method for calculating the total loss of the discriminator, which has two input parameters: prediction of the discriminator on the real (target) image and prediction on the fake (generated) image. The total loss of the discriminator is the sum of the binary cross-entropy loss of the real images and an array of ones (using *ones_like()*), and the binary cross-entropy loss of the generated images and an array of zeros (using *zeros_like()*). The *BinaryCrossentropy()* function is used again to calculate these losses.

The second abstract method that the *Discriminator* class implements is the method for creating the model. Since the discriminator has two images on the input, it is necessary to concatenate these inputs first. The *Concatenate* layer is used for this. This layer is followed by a series of downsampling blocks, which were created using the prepared method in the same way as for the generator. In this way, the final structure of the discriminator proposed in the previous chapter is achieved.

ConGAN

This section describes how the generator and discriminator are interconnected using the *ConGAN* class. Its purpose is that these two models are trained simultaneously by an adversarial process. The generator learns how to create realistic-looking images, while the discriminator learns how to distinguish real images from generated ones. When this class is initialized, a log directory and instances of the *Checkpoint* and *SummaryWriter* classes are created. The log directory is a directory in which the data created during the training will be stored.

The *Checkpoint* class is used to save and restore models, which can be helpful in case of interruption of a long running training task. A checkpoint saves a graph of dependencies between Python objects, such as layers and optimizers, with named edges, and this graph is used to match variables when restoring a checkpoint. A checkpoint is also used by the generator, where, before generating synthetic images, the generator is initialized with values from that checkpoint. The generator and discriminator, together with their optimizers, are passed as input parameters of the checkpoint so that they can be monitored and stored.

The *SummaryWriter* class is used to record the losses of individual models, which can then be examined. It is an interface representing a stateful summary writer object. The output of this class can be easily viewed in the TensorBoard visualization tool to monitor the progress of the training.

The core of ConGAN is the method for training these models. Its input parameters are the total number of epochs, the training dataset, the testing dataset, the output period and the checkpoint period. At the beginning of each epoch, one image is generated and saved. The testing dataset is used to generate this image. This allows us to see what the generated images look like in each epoch. The generator method, already described in this section, is used for the image generation itself. How often individual images are generated is regulated by the input argument of this method – the output period.

The generation is followed by the training process itself. It iterates over the training dataset, and a training step is taken for each batch. One training step consists of the following parts:

1. For each example input, an output is generated.

2. The discriminator receives the input image and the generated image, from which it makes a prediction about the generated image. It then receives the input image and the target image, from which it makes a prediction about the target image.

3. Based on these predictions, the losses of the generator and discriminator are then calculated using the implemented methods of these models.
4. Gradients are calculated from these losses. The *GradientTape* class is used for this purpose. It records operations for automatic differentiation. The gradients of loss are calculated with respect to both the generator and the discriminator variables (trainable and non-trainable).
5. These gradients are applied to the optimizer, using the implemented method for updating weights, which ensures that the gradients are applied to the variables.
6. Lastly, these losses are logged using *SummaryWriter*.

After completing the training step, a checkpoint is saved. The checkpoint period can also be regulated by the input parameter of the training method – the checkpoint period.

During the training, information about the status of the training is printed to the console, such as the epoch number, how many training steps are completed, and how long the epoch lasted. After the training is completed, the last checkpoint is saved, and summary statistics for the given training process are printed, containing the total training time and the average time per epoch.

5.4 Source Code Structure and Usage

The source code is divided into several files. Their hierarchy is given below:

- `pretrained/`
    - `chasedb1/`
    - `combined/`
    - `drive/`
    - `hrf/`
    - `stare/`
- `srig/`
    - `__init__.py`
    - `congan.py`
    - `data_loader.py`
    - `discriminator.py`
    - `generator.py`
    - `model.py`
- `requirements.txt`
- `srig.py`

The `pretrained` directory contains subdirectories with checkpoints where the weights of pretrained models are stored. These checkpoints are used to initialize the retinal image generator. The `srig` directory contains the source code for each of the implemented classes described in the previous sections. The `__init__.py` file is used to mark the `srig` directory as the Python package directory from which the classes are imported. The `requirements.txt` file contains a list of libraries that were used during the implementation. These libraries are required to run the program. The last file, `srig.py`, represents the entry point of the program where the input arguments are processed. Based on these arguments, further execution of the program takes place. The individual arguments are described later in this section.

Installation

The implemented program requires, in addition to the libraries used, **Python 3** installed along with **PIP** and **tkinter**. PIP is the package manager for Python packages, and tkinter is the standard graphical user interface (GUI) for Python. The PIP is used to install all the required libraries so that the user does not have to install them one by one. A list of these libraries is given in the *requirements.txt* file.

To install them, one of the following commands can be used:

- `pip install -r requirements.txt`
- `python3 -m pip install -r requirements.txt`

This will automatically install all the libraries listed in that file. The program is then ready for use.

Program Arguments

The program can do three different things: prepare training data, train the model, or generate retinal images. These operations are mutually exclusive, and only one of them can be performed while the program is running. Therefore, the program has a required argument in which one of the three options is selected, and then further arguments to the corresponding option can be specified. All supported arguments are listed below.

Mutually exclusive arguments

`--prepare`	Prepare input data for training.
`--train`	Train the model.
`--generate`	Generate retinal images.

Data preparation arguments

`--bv_dir BV_DIR`	Specify a directory with images of blood vessels.
`--retina_dir RET_DIR`	Specify a directory with retinal images.

Training arguments

`--train_dir TRAIN_DIR`	Specify a directory with training images.
`--test_dir TEST_DIR`	Specify a directory with testing images. The default value is the training directory.
`--log_dir LOG_DIR`	Specify the log directory. The default value is `./logs`.
`--checkpoint_period C_PERIOD`	Specify the period of checkpoint storing. The default value is 1. Negative values disable the checkpoints.
`--output_period O_PERIOD`	Specify the period of output image generation. The default value is 1. Negative values disable the generation.
`--epochs EPOCHS`	Specify the number of epochs. The default value is 1.
`--batch_size BATCH_SIZE`	Specify the batch size. The default value is 1.

Generation arguments

`--input INPUT`	Specify the input file or directory.
`--checkpoint_dir CHCKP_DIR`	Specify the checkpoint directory.
`--display_output`	Display generated output.

Common arguments

`-h, --help`	Display the help.
`--output_dir OUT_DIR`	Specify the output directory. The default value is `./output`.

Examples

To illustrate, below are examples of how to run this program.

- `python srig.py --prepare --bv_dir blood_vessels/ --retina_dir retinas/ --output_dir training_data/`

  This processes images from the `blood_vessels` and `retinas` directories, prepares them for training and saves the resulting images in the `training_data` directory.

- `python srig.py --train --train_dir training_data/ --epochs 100`

  This trains the model for 100 epochs using images in the `training_data` directory. On each epoch, it generates an output image to show its progress, and saves a checkpoint with the current weights. All generated data are stored in the default directory, `logs`.

- `python srig.py --train --train_dir training_data/ --test_dir testing_data/ --log_dir summary/ --epochs 100 --output_period 10 --checkpoint_period 20`

  This trains the model for 100 epochs using images in the `training_data` directory. It generates an output image every 10 epochs and saves a checkpoint every 20 epochs. It uses images from the `testing_data` directory to generate output images. All generated data are stored in the `summary` directory.

- `python srig.py --generate --input examples/image.jpg --display_output`

  This generates an image of the retina using `image.jpg` as input, saves it in the default directory, `output`, and displays the generated image on the screen.

- `python srig.py --generate --input examples/ --checkpoint_dir output/checkpoints/ --output_dir generated_retinas/`

  This generates as many retinal images as there are images in the `examples` directory. Images are generated using the model with weights initialized from a checkpoint from the `output/checkpoints` directory. The generated images are saved in the `generated_retinas` directory.

Chapter 6

Training and Testing

This chapter focuses on the actual training of the model, which was proposed in Chapter 4 and subsequently implemented in Chapter 5. It contains a description of what data was used to train this model, a description of the training process itself, along with problems that occurred during the training and how I solved them. Finally, an evaluation of the obtained results is given.

6.1 Data Source

An important basis for training and subsequent testing of the algorithm for automatic generation of synthetic retinal images is to have a sufficiently large number of retinal images. These images should form a representative set of retinas. From publicly available databases of retinas such as ADCIS [15], CHASE_DB1 [31], DRIVE [12], FIRE [22], HRF [10] and STARE [18]. Only those databases that already contain professionally annotated blood vessels were used.

Those used databases include the CHASE_DB1 database, which consists of 84 color digital images with a resolution of 999×960 pixels. 28 of these images are retinal images and 56 images of blood vessels. Each retina has two corresponding images of segmented blood vessels that differ in the number of visible blood vessels. Therefore, it is possible to create 56 pairs, where an example of one such retina is given in Figure 6.1 below.

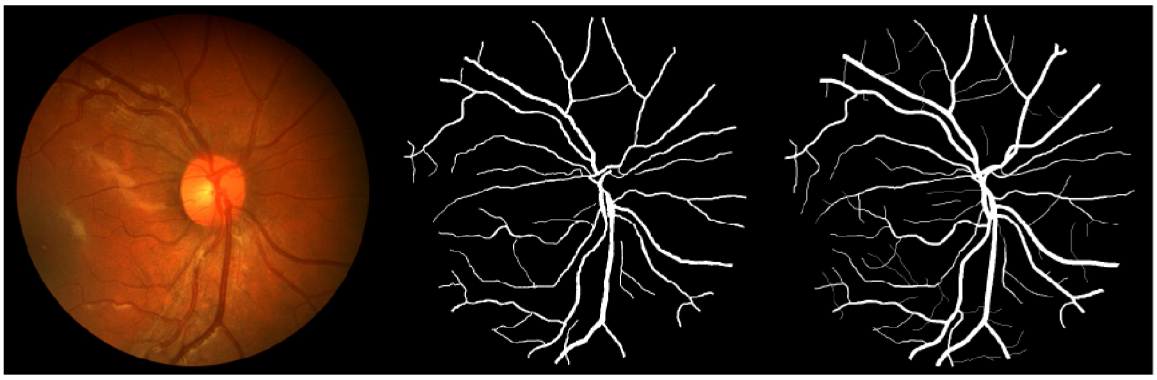


Figure 6.1: Example of a retina from the CHASE_DB1 database. The two images with segmented blood vessels differ slightly in the number of visible blood vessels.

Another database used is the DRIVE database, which contains 100 images with a resolution of 565×584 pixels. This database already divides these images into a training and testing set, where each contains 20 different retinas. For each retina, there is one image of the retina itself and one image with a background mask. Only the training set contains the required segmented blood vessels. Therefore, only 20 pairs of images can be created from this database.

The HRF database is also divided into retinal images, corresponding background masks and segmented blood vessels. A total of 45 different images of retinas with a resolution of 3504×2336 pixels are available in this database, which is the highest resolution of all databases.

The last database used is the STARE database. It contains a total of 397 different images of retinas, but only 20 are hand-labeled images by an expert, so only 20 pairs were created. The resolution of individual images is 700×605. Other databases were not used because they do not contain segmented blood vessels.

A total of 141 pairs of images (retinas and their segmented blood vessels) are available. Since the input resolution of the system is 1024×1024 and the images from the DRIVE and STARE databases have a much lower resolution, these images were enlarged using an online tool[1] that uses a neural network with an algorithm adjusted for images, thus making the enlarging process in high quality. Details on individual sets of images are given in Table 6.1 below.

Database	**Resolution**	**Pairs**
CHASE_DB1	999×960	56
DRIVE	1130×1168	20
HRF	3504×2336	45
STARE	1400×1210	20
	$\sum$	141

Table 6.1: Used databases that are publicly available.

6.2 Training

In order for the generator to generate retinal images, it must first be trained. The training loop begins with the generator receiving a black and white image of blood vessels. This image is used to create an image of the retina. The discriminator then classifies real images (drawn from the training set) and fake images (produced by the generator). The loss is calculated for each of these models in order to calculate the gradients used to update the generator and discriminator weights. This is the basic concept of how the combined model is trained.

Monitoring

The implemented training loop saves the logs of each epoch, so they can be easily viewed in the TensorBoard tool to monitor the progress of the training. For this purpose, a separate TensorBoard process can be started using the following command:

```
tensorboard --logdir [log_dir]
```

[1] https://bigjpg.com/

where `log_dir` is a directory containing the training logs. The default directory with this data is *logs/summary* in the current working directory.

The loss during training typically reduces fairly quickly during the first few training iterations, and then the reduction slows down because the learning algorithm performs small changes to find the exact local minimum. This is shown in Figure 6.2. The value of $ln(2)$, which is approximately 0.69, is a good reference point for these losses, as it indicates a perplexity of 2 – that the discriminator is, on average, equally uncertain about the two options. For the discriminator loss, a value below 0.69 means the discriminator is doing better than random on the combined set of real and generated images. For the generator loss, a value below 0.69 means the generator is doing better than random at fooling the discriminator.

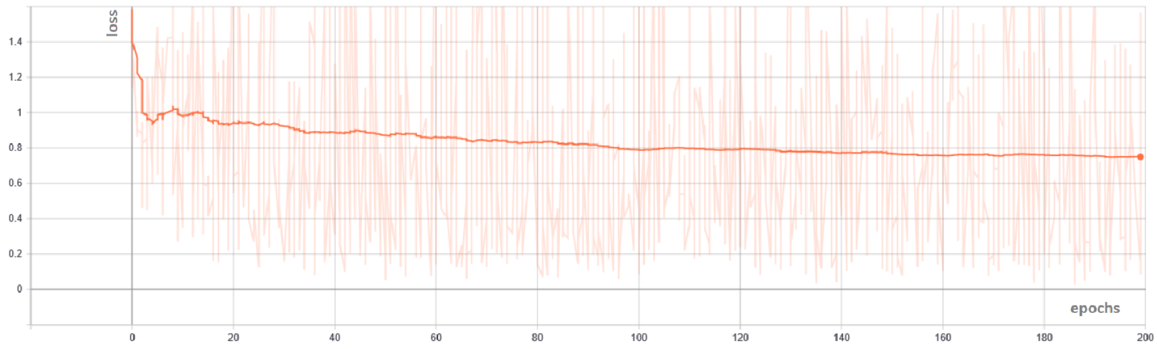


Figure 6.2: Loss of the discriminator during the 200 epochs. The horizontal axis shows epochs, vertical axis losses. The values are smoothed for better clarity.

The generator and discriminator are trained simultaneously. It is important that the generator and discriminator do not overpower each other. When training a GAN, it is therefore necessary to monitor the loss of individual models. If either the loss of the generator or the loss of discriminator gets very low, it indicates that this model is dominating the other, and the combined model is not being successfully trained. This situation is shown in Figure 6.3. In order to solve this, the learning rate of the model that dominates must be reduced or, conversely, the learning rate of the model that is dominated must be increased.

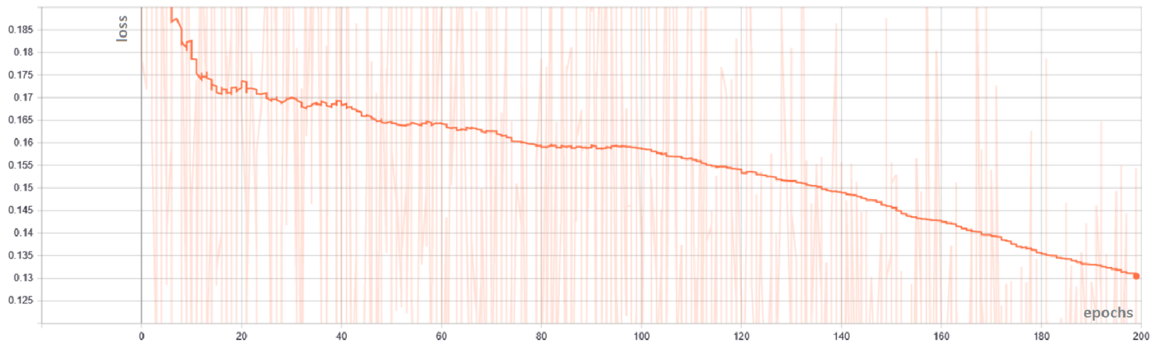


Figure 6.3: Loss of the discriminator that dominates the generator. The horizontal axis shows epochs, vertical axis loss. The values are smoothed for better clarity.

Training Duration

The training of the combined model requires that the algorithm runs over the entire training set many times, with the weights changing as the model makes errors in each iteration. The problem is how to decide when to stop learning. It is not desirable to stop training until the local minimum has been found, but training too long leads to overfitting of the model. There are many ways to solve this problem, but the most obvious ones are not sufficient. One of these solutions is to set some predefined number of iterations and train until that is reached, but this poses a risk that the model will be overfitted by then, or not learned enough. Another solution is to stop only when some predefined minimum loss is reached, but that might mean the algorithm never terminates, or that it overfits.

This is where the testing set comes in useful. The model is trained for a predetermined period of time, and then the testing set is used to estimate how well the model can generalize. The training is carried on for a few more iterations, and the whole process is repeated. As training progresses, the generated images will look increasingly real. The first image generated by an untrained generator contains mostly noise (Figure 6.4), but thanks to the architecture of the generator, where generation depends on the input image of blood vessels, a slight structure of blood vessels can already be seen in the generated image. After just one epoch, the generated image resembles a real retina, where this retina begins to acquire its shape and color. Such an image is shown in Figure 6.5.

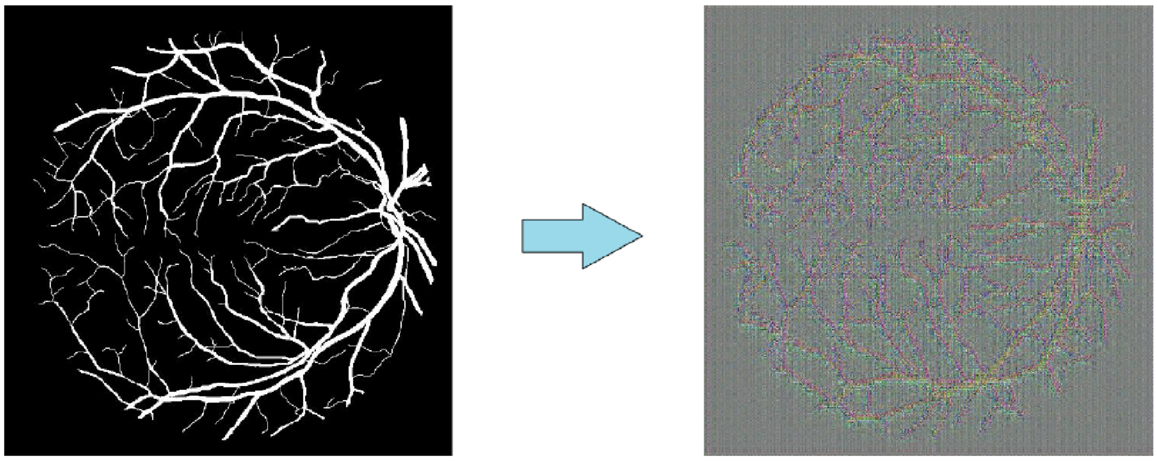


Figure 6.4: An image that was created by an untrained generator from an input image of blood vessels.

Therefore, I approached such a training solution, where at the beginning of each epoch, an image from the testing set is generated. These images can be used to monitor the generalization capability of the model at its current stage of learning. Based on the quality of these images, a decision when to stop training the model is made. Another indicator of when to stop training is monitoring the loss of both the generator and the discriminator. The training can be terminated once the error stops decreasing. At some stage the error on the testing set will start increasing again, because the model has stopped learning about the function that generated the data, and started to learn about the noise that is in the data itself. At this stage the training is stopped.

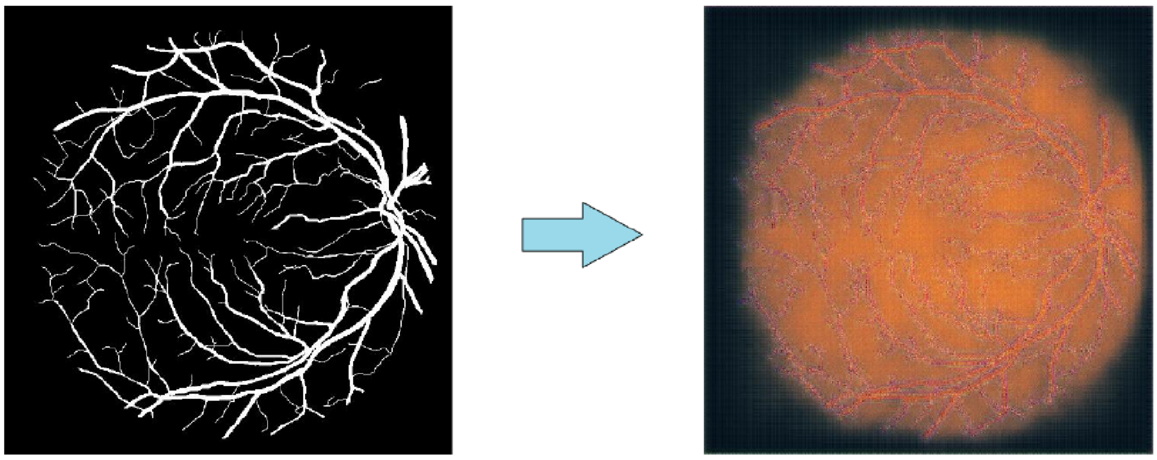


Figure 6.5: An image that was created by a generator from an input image of blood vessels. The generator was trained for only one epoch.

Model Parameters

Building machine learning models requires a selection of various parameters of these models, such as the dropout rate in a layer or the learning rate. These decisions impact the metrics of the model. Therefore, an important step in the machine learning workflow is to identify the best parameters for a given problem.

The final parameters of the model were obtained by experimenting with different values of these parameters and their combinations, where the values that produced the best results were selected. Details on each parameter are given below. The number of epochs varied for each database and their exact values are given in Section 6.3.

- **Weights** are the first parameter that was set. The weights are initialized to small random numbers, both positive and negative. Random values in this range were used so that the learning starts from different places for each run. These values are also about the same size, as it is desirable for all of the weights to reach their final values at about the same time. To initialize the weights in this way, a normal (or Gaussian) distribution was used. A normal distribution is a type of continuous probability distribution for a real-valued random variable. The general form of its probability density function is [48]:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2} \tag{6.1}$$

  where the parameter μ is the mean of the distribution, and σ is its standard deviation. In this case, the mean is set to 0 and the standard deviation is set to 0.025. Resulting distribution can be seen in Figure 6.6.

- **Learning rate** is a parameter that indicates how much these weights will change each iteration. It was necessary to choose it so that both models are trained at a similar rate. Thanks to this, there will be no situation where one of the models dominates the other. If one model dominates the other, the model as a whole gives poor results. For the generator the learning rate 0.00025 was chosen, and for the discriminator 0.0003 was chosen.

- **Dropout** layers are located only in the decoder part of the generator. During training, they randomly drop out a number of output values of the layer, which helps prevent overfitting. The exact number is given by the dropout rate, which is a fraction of the values that are dropped out. The dropout rate of these layers was set to the value of 0.5 (half of all values were dropped), but only in the first three upsampling layers.
- **Batch size** is the number of training examples utilized in one iteration. With small values the model converges quickly at the cost of noise in the training process. The smaller the batch the less accurate the estimate of the gradient will be. It also requires less memory, since the model is trained using fewer samples at once. On the other hand, with large values it converges slowly, but with more accurate estimates of the gradient. Based on the available data and computational power, I was only able to use a value of 2 as the batch size.
- **Number of filters** differs for individual convolution or deconvolution layers. Convolutional neural networks do not learn a single filter. They learn multiple features in parallel for a given input. This gives the model many different ways of extracting features from an input. In this case, I started with 32 filters for the first layer and gradually doubled this number up to 512 filters. The last deconvolution layer uses 3 filters in the case of the generator, which correspond to the individual color channels of the output image. In the case of the discriminator, the last convolution layer uses only one filter, thanks to which the required 32×32×1 patches are obtained.
- **Filter size** represents the size of each filter. In the generated images, I came across checkerboard artifacts caused by unequal coverage of the pixel space in the generator, as shown in Figure 6.7. To solve this problem, a filter size that is divisible by the stride size was used. The selected value is therefore 4×4 pixels. This value is used wherever convolution and transposed convolution layers are used, i.e., in the generator and discriminator.

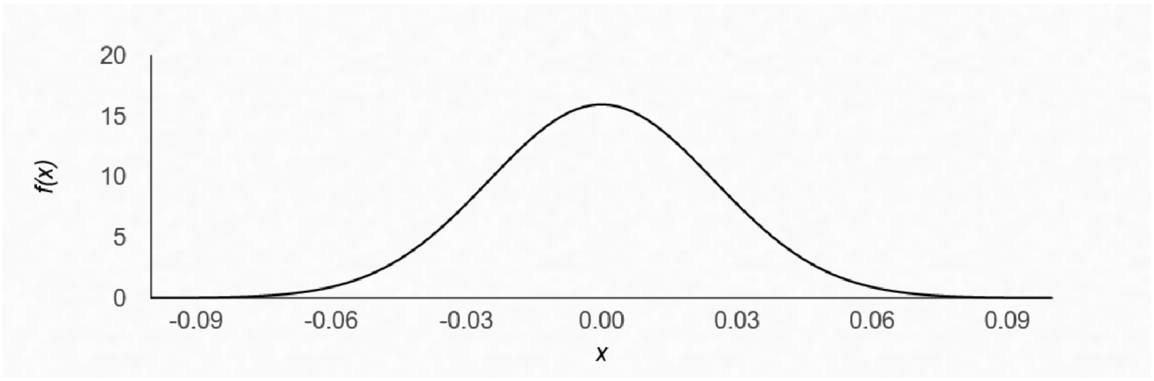


Figure 6.6: Probability density function of a normal distribution with the mean value of 0 and the standard deviation of 0.025.

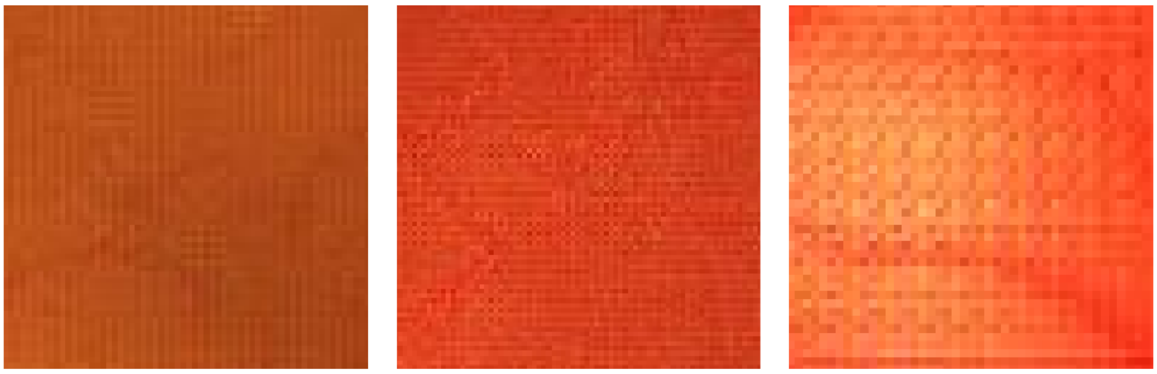


Figure 6.7: Checkerboard artifacts caused by mismatching strides and filter sizes.

Final Training of the Model

The generator model contains a total of 13,184 fixed parameters that do not change during training, and over 67,000,000 trainable parameters, the values of which need to be adjusted by learning. The discriminator model has less than 700,000 parameters in total.

The learning itself was performed on individual databases and then one learning was performed on images from all databases together. Some images are of poor overall quality or contain retinas with various diseases or other damage, so these images were excluded from training sets so that the generator learns to generate only high quality images of healthy retinas. Testing sets were created from the excluded images, which were used to monitor how well the generator generalizes. A different number of epochs was used for each database, as each database contains a different number of images. More detailed information about individual trainings is given in Table 6.2, which contains information about the number of images in each set, the total number of epochs and how long the given training lasted. In this way, several files containing learned weights for individual databases and their combinations were created and used to generate new retinal images.

Database	**Training Images**	**Testing Images**	**Epochs**	**Time (minutes)**
CHASE_DB1	42	14	24	29
DRIVE	16	4	18	12
HRF	32	13	37	42
STARE	12	8	21	11
All combined	102	39	41	139

Table 6.2: Training on individual databases.

6.3 Evaluation

Using the implemented and trained generator, several databases of synthetic retinal images were created, one for each database on which the generator was trained. This generator was able to generate new images of retinas in a resolution of 1024×1024 pixels. The original databases together provided 141 input images with segmented blood vessels. To increase the number of input images from which retinal images were generated, the original images were flipped horizontally, vertically, and both horizontally and vertically. In this way, three new images were created from each input image that produce a slightly different output. Thus, 564 input images were available for the generation process. A total of 2,820 retinal images were created from five different sets of learned weights. These newly created retinas were subsequently evaluated.

The person evaluating the generated data does not need to have the expertise of an ophthalmologist to be able to assess whether a given retina is real or not. It is only sufficient if he is an informed layman familiar with the subject. Thanks to this, I was able to perform this evaluation myself without the need for the assistance of an ophthalmologist.

The generalization capabilities of the model are at a high level for individual databases, where Figure 6.8 shows an example of a newly generated image, as well as the original image and the corresponding blood vessels for comparison. Some of the generated images are indistinguishable from the real ones, which was the goal of this thesis.

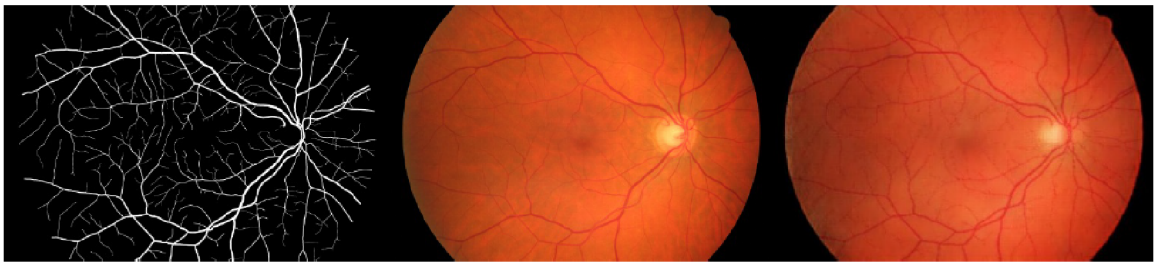


Figure 6.8: Example of a generated image. The image on the left is the input, the image in the middle is the original, real image, and the image on the right is the synthetic image.

Despite the fact that the newly generated images are similar to the original ones, as they have the same bloodstream, it can be said that they are new retinas. The new images have different colors, the optic disc and fovea are different, and they also differ in small structures on the retina. Thus, this generator allows to generate an image of a healthy retina from the blood vessels of a damaged retina, as shown in Figure 6.9, or from an overexposed image to an image with a normal exposure that is less bright. An example of an overexposed image is shown in Figure 6.10.

However, if the images from different databases are combined with each other, such quality is no longer achieved. For example, the CHASE_DB1 database consists of retinal images that are centered on the optic disc. As a result, its position and shape were overfitted, which was reflected in images from other databases. An example of such a case is shown in Figure 6.11. The generator trained on combination of all databases had the best generalization capability because they contained a large number of different images. These images were taken from different angles and captured differently sized parts of the retina. On the contrary, the DRIVE and STARE databases achieved the worst results, as they were trained on a small number of images. For a higher quality of images from different databases of blood vessels, it would be necessary to have a larger number of input images on which the model would be trained.

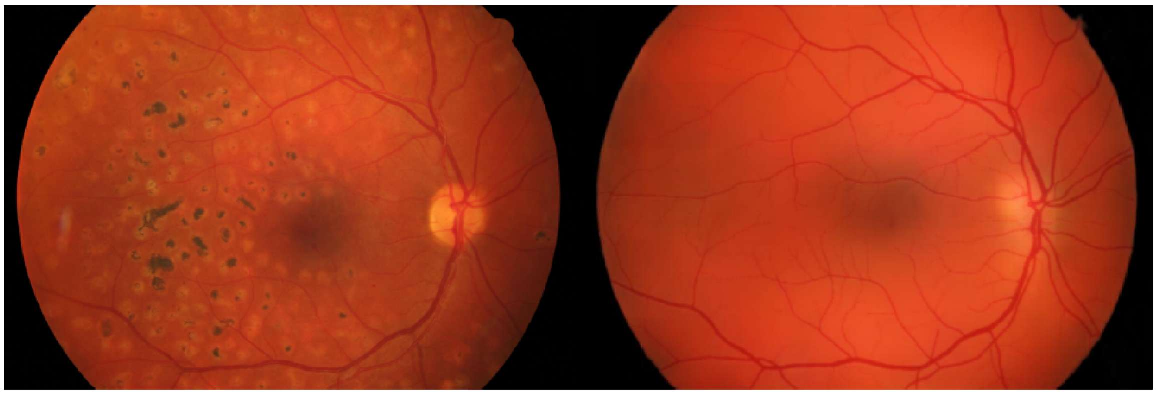


Figure 6.9: Example of a generated image. The image on the left is an image of a damaged retina and the image on the right is the image generated from the bloodstream of the retina on the left.

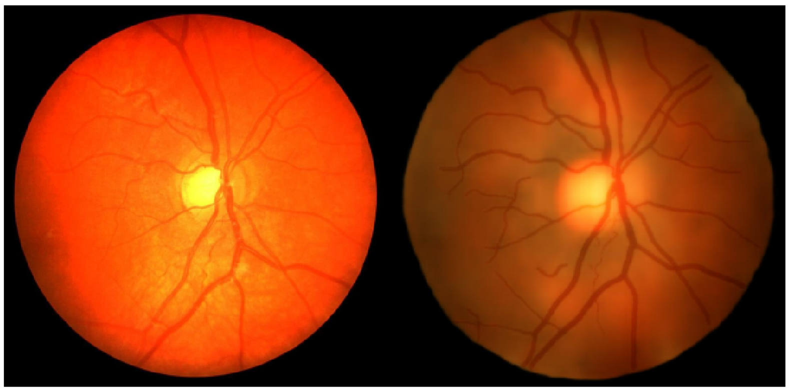


Figure 6.10: On the right is an example of an image generated from the bloodstream of the retina on the left, which is overexposed.

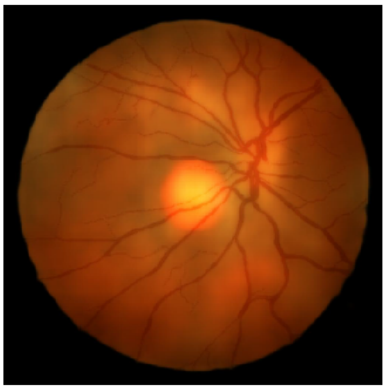


Figure 6.11: Example of an image generated by the generator trained on the CHASE_DB1 database. The optic disc is overfitted to the center of the retina. The input blood vessels for this image are from the DRIVE database.

Chapter 7

Conclusion

Human eyes provide us with vision, and the most important part of the human eye is the retina. The retina is also the most sensitive part of the human eye, and therefore diseases of the retina can lead to vision loss. For this reason, it is important to protect our eyesight, as its loss means a significant deterioration in our quality of life. Special equipment, a fundus camera, is needed to capture the retina, so it is not an easy task to obtain these images in large quantities. Therefore, the main objective of this work was to design and implement a system that would be able to generate new synthetic images of retinas that would extend existing databases. These images could then be used as a learning tool for ophthalmologists to practice their knowledge or for the development of medical or biometric systems.

This thesis provides a theoretical basis for the anatomy of the human eye and some selected retinal diseases. From a technical point of view, it focuses on various methods of machine learning and on the basic principles of neural networks. Specifically, it focuses on deep learning, which includes specific types of neural networks, namely, convolutional and generative adversarial networks, based on which a method of generating synthetic retinal images was proposed. The thesis provides a detailed description and implementation of this solution and eventually describes the process of training the proposed system. After the system was trained, a database of synthetic retinal images was created, the quality of which was subsequently assessed.

The proposed solution uses an image-to-image translation, where the system is provided with a black and white image at the input containing only bloodstream, on the basis of which a color image of the entire retina is generated. The system consists of two neural networks, one of which is a generator that generates retinal images from an input image of blood vessels, and the other is a discriminator that has two images at the input – the same image of blood vessels and a corresponding retinal image that is classified as real or synthetic. In order for this system to be able to generate realistic-looking images of retinas, it had to be trained first. The training was performed on several publicly available databases, which together provided 141 input images. The generator and discriminator were trained simultaneously, where the aim was for the generator to produce images of such a quality that the discriminator would not be able to distinguish them from real ones, and at the same time that the discriminator had its classification capability at the highest possible level. Care had to be taken to ensure that one model did not dominate the other, as the system as a whole would produce poor results. It was therefore necessary to find an equilibrium between these models.

After the system has been trained, there is no need for the discriminator and only the trained generator is used. With this generator, a database of over 2,800 synthetic images in a resolution of 1024×1024 pixels was created from the available data. Many of the images generated in this way were indistinguishable from real images of the retina, which was the aim of this work. Despite the fact that these generated images are based on the bloodstream of real retinas, they can be considered as images of new retinas, as, except for the given blood vessels, they differ in everything else, such as color, optic disk and fovea. In addition, the generator was trained on a set of healthy retinas, so it is possible, for example, to generate a healthy retina from the bloodstream that belongs to a retina with a disease.

Future work should focus on training this network on a larger number of retinal images from different databases. This network can be extended to generate images in much higher resolution, but it would be necessary to provide sufficient computing power and also to provide input images that are in at least as high a resolution as the desired output resolution of the generator. Another possible continuation of this work would be to design and implement a system that would generate black and white images of segmented retinal blood vessels from random noise. The output of that system would then be connected to the input of the generator from this thesis. In this way, it would be possible to generate completely new retinas that would not depend on the input data, as the input data would be random noise.

Bibliography

[1] Alpaydin, E. and Bach, F. *Introduction to Machine Learning*. 2nd ed. Cambridge, London: MIT Press, 2014. ISBN 978-0-262-02818-9.

[2] Amidi, A. and Amidi, S. *Leaky ReLU function*. 2018. [Online; visited 14.03.2020]. Available at: https://stanford.edu/~shervine/teaching/cs-229/illustrations/leaky-relu.png.

[3] Amidi, A. and Amidi, S. *Neural network architecture*. 2018. [Online; visited 14.03.2020]. Available at: https://stanford.edu/~shervine/teaching/cs-229/illustrations/neural-network-en.png.

[4] Amidi, A. and Amidi, S. *ReLU function*. 2018. [Online; visited 14.03.2020]. Available at: https://stanford.edu/~shervine/teaching/cs-229/illustrations/relu.png.

[5] Amidi, A. and Amidi, S. *Sigmoid function*. 2018. [Online; visited 14.03.2020]. Available at: https://stanford.edu/~shervine/teaching/cs-229/illustrations/sigmoid.png.

[6] Amidi, A. and Amidi, S. *Tanh function*. 2018. [Online; visited 14.03.2020]. Available at: https://stanford.edu/~shervine/teaching/cs-229/illustrations/tanh.png.

[7] Arbib, M. A. *The Handbook of Brain Theory and Neural Networks*. 2nd ed. Cambridge, London: MIT Press, 2003. ISBN 0–262–01197–2.

[8] Aubrecht, T. *Detekce onemocnění ve snímku sítnice oka*. Brno, 2017. 40 p. Bachelor's thesis. Vysoké učení technické v Brně. Fakulta informačních technologií. Vedoucí práce Ing. Lukáš Semerád.

[9] Beneš, P. *Přístroje pro optometrii a oftalmologii*. 1st ed. Brno: Národní centrum ošetřovatelství a nelékařských zdravotnických oborů, 2015. ISBN 978-80-7013-577-8.

[10] Budai, A., Bock, R. et al. *High-Resolution Fundus Image Database*. 2013. [Online; visited 10.03.2020]. Available at: https://www5.cs.fau.de/research/data/fundus-images/.

[11] Calhoun, J. S. *Retinal Hemorrhage*. 2013. [Online; visited 20.11.2019]. Available at: https://imagebank.asrs.org/file/10070/retinal-hemorrhage.

[12] Challenge, G. *DRIVE: Digital Retinal Images for Vessel Extraction*. 2012. [Online; visited 10.03.2020]. Available at: https://drive.grand-challenge.org/.

[13] Charniak, E. *Introduction to Deep Learning*. 1st ed. Cambridge, London: MIT Press, 2018. ISBN 978-0-262-03951-2.

[14] Chollet, F. *Deep learning with Python*. 1st ed. Shelter Island, NY: Manning, 2018. ISBN 978-1-61729-443-3.

[15] Decencičre, E. *TeleOphta: Machine learning and image processing methods for teleophthalmology*. 2013. [Online; visited 10.03.2020]. Available at: http://www.adcis.net/en/Download-Third-Party/E-Ophtha.html.

[16] Deshpande, M. *Perceptron scheme*. 2017. [Online; visited 20.02.2020]. Available at: https://pythonmachinelearning.pro/wp-content/uploads/2017/09/Single-Perceptron.png.

[17] Fortuner, B. *Cross-Entropy*. 2017. [Online; visited 22.3.2020]. Available at: https://ml-cheatsheet.readthedocs.io/en/latest/_images/cross_entropy.png.

[18] Goldbaum, M. *STARE: Structured Analysis of the Retina*. 2004. [Online; visited 10.03.2020]. Available at: http://cecas.clemson.edu/~ahoover/stare/.

[19] Goodfellow, I., Bengio, Y. and Courville, A. *Deep Learning*. 1st ed. Cambridge, MA: MIT Press, 2016. ISBN 978-0-262-03561-3.

[20] Goodfellow, I. J., Pouget Abadie, J., Mirza, M., Xu, B., Warde Farley, D. et al. *Generative Adversarial Networks*. 2014. [Online; visited 18.03.2020]. Available at: https://arxiv.org/abs/1701.00160.

[21] Hamel, C. *Fundus of patient with retinitis pigmentosa*. 2006. [Online; visited 20.11.2019]. Available at: https://en.wikipedia.org/wiki/Retinitis_pigmentosa#/media/File:Fundus_of_patient_with_retinitis_pigmentosa,_mid_stage.jpg.

[22] Hernandez Matas, C., Zabulis, X. et al. *FIRE: Fundus Image Registration Dataset*. 2017. [Online; visited 10.03.2020]. Available at: http://www.ics.forth.gr/cvrl/fire/.

[23] Hrvolová, B. *Biofyzika*. 1st ed. Ostrava: Vysoká škola báňská - Technická univerzita Ostrava, 2013. ISBN 978-80-248-3105-3.

[24] Hycl, J. and Trybučková, L. *Atlas oftalmologie*. 2nd ed. Praha: Triton, 2008. ISBN 978-80-7387-160-4.

[25] Isola, P., Zhu, J.-Y., Zhou, T. and Efros, A. A. *Image-to-Image Translation with Conditional Adversarial Networks*. 2016. [Online; visited 21.03.2020]. Available at: https://arxiv.org/abs/1611.07004.

[26] Jain, A. K. et al. *Introduction to Biometrics*. 1st ed. Spring Street, New York: Springer, 2011. ISBN 978-0-387-77325-4.

[27] Kingma, D. P. and Ba, J. *Adam: A Method for Stochastic Optimization*. 2014. [Online; visited 01.04.2020]. Available at: https://arxiv.org/abs/1412.6980.

[28] Kolář, P. et al. *Věkem podmíněná makulární degenerace*. 1st ed. Praha: Grada, 2008. ISBN 978-80-247-2605-2.

[29] Kvapilíková, K. *Anatomie a embryologie oka*. 1st ed. Brno: Institut pro další vzdělávání pracovníků ve zdravotnictví, 2000. ISBN 80-7013-313-9.

[30] LARSEN, A. B. L., SØNDERBY, S. K., LAROCHELLE, H. and WINTHER, O. *Autoencoding beyond pixels using a learned similarity metric*. 2015. [Online; visited 21.03.2020]. Available at: https://arxiv.org/abs/1512.09300.

[31] LONDON, K. U. *Retinal Image Analysis*. 2016. [Online; visited 10.03.2020]. Available at: https://blogs.kingston.ac.uk/retinal/chasedb1/.

[32] MARSLAND, S. *Machine Learning, An Algorithmic Perspective*. 1st ed. Boca Raton, FL: CRC Press, 2009. ISBN 978-1-4200-6718-7.

[33] MAZINANI, S. R. *Development of novel organic optoelectronic technologies for biomedical applications*. Saint-Étienne, France, 2017. 93 p. Dissertation. l'Ecole des Mines de Saint-Étienne.

[34] MEDISAVE.CO.UK. *Direct ophthalmoscope*. 2017. [Online; visited 11.11.2019]. Available at: https://www.medisave.co.uk/heine-mini3000-2-5v-ophthalmoscope.html.

[35] MF.CZ. *Geografická atrofie*. 2016. [Online; visited 20.11.2019]. Available at: https://img.mf.cz/060/617/b.jpg.

[36] MF.CZ. *Vlhká forma s edémem*. 2016. [Online; visited 20.11.2019]. Available at: https://img.mf.cz/062/617/c.jpg.

[37] NEI. *Retinitis Pigmentosa* [online]. 2014. Updated July 10, 2019 [cit. 20. November 2019]. Available at: https://www.nei.nih.gov/learn-about-eye-health/eye-conditions-and-diseases/retinitis-pigmentosa.

[38] NIDEK INTL.COM. *Fundus camera*. 2015. [Online; visited 11.11.2019]. Available at: http://www.nidek-intl.com/product/ophthaloptom/diagnostic/dia_retina/afc-330.html.

[39] NVIDIA. *GPU-Accelerated TensorFlow* [online]. 2018 [cit. 16. February 2020]. Available at: https://www.nvidia.com/en-sg/data-center/gpu-accelerated-applications/tensorflow.

[40] PATHAK, Y. *Artificial Neural Network for Drug Design, Delivery and Disposition*. 1st ed. Cambridge, US: Academic Press, 2015. ISBN 9780128017449.

[41] PATTANAYAK, S. *Pro deep learning with TensorFlow : a mathematical approach to advanced artificial intelligence in Python*. 1st ed. New York: Manning, 2017. ISBN 978-1-4842-3095-4.

[42] PAŠTA, J. *Základy očního lékařství*. 1st ed. Praha: Univerzita Karlova, Nakladatelství Karolinum, 2017. ISBN 978-80-246-2460-0.

[43] RHCASTILHOS. *Schematic diagram of the human eye*. 2007. [Online; visited 11.11.2019]. Available at: https://en.wikipedia.org/wiki/Human_eye#/media/File:Schematic_diagram_of_the_human_eye_en.svg.

[44] RONNEBERGER, O., FISCHER, P. and BROX, T. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. 2015. [Online; visited 21.03.2020]. Available at: https://arxiv.org/abs/1505.04597.

[45] Sphinx Gallery. *Underfitting vs. Overfitting*. 2019. [Online; visited 16.3.2020]. Available at: https://scikit-learn.org/stable/_images/sphx_glr_plot_underfitting_overfitting_001.png.

[46] WelchAllyn.com. *Indirect ophthalmoscope*. 2017. [Online; visited 11.11.2019]. Available at: https://www.welchallyn.com/en/products/categories/physical-exam/eye-exam/ophthalmoscopes--binocular-indirect/binocular_indirect_ophthalmoscope.html.

[47] Wikipedia. *Cross entropy* [online]. 2020 [cit. 22. March 2020]. Available at: https://en.wikipedia.org/wiki/Cross_entropy.

[48] Wikipedia. *Normal distribution* [online]. 2020 [cit. 12. April 2020]. Available at: https://en.wikipedia.org/wiki/Normal_distribution.

[49] Wong, G. *Retina of the human eye*. 2016. [Online; visited 11.11.2019]. Available at: https://d3b3by4navws1f.cloudfront.net/177901919.jpg.

[50] Wong, T. Y. and Ting, D. S. W. Generative Adversarial Networks (GANs) for Retinal Fundus Image Synthesis. In: Carneiro, G. and (Eds.), S. Y., ed. *Computer Vision-ACCV 2018 Workshops: 14th Asian Conference on Computer Vision*. Perth, Australia: Springer, 2019, p. 289–302. ISBN 978-3-030-21073-1.

[51] Yegulalp, S. *What is TensorFlow? The machine learning library explained* [online]. 2019 [cit. 14. February 2020]. Available at: https://www.infoworld.com/article/3278008/what-is-tensorflow-the-machine-learning-library-explained.html.

[52] Zhang, Y.-Q. and Rajapakse, J. C. *Machine Learning and Biometrics*. 1st ed. Hoboken, N.J.: Wiley, 2008. ISBN 978-0-470-11662-3.

Appendix A

Contents of the Attached DVD

The directory structure on the attached DVD is shown and described below.

- **databases**
  - **real** – contains individual databases of real retinal images
  - **synthetic** – contains individual databases of synthetic retinal images
- **datasets** – prepared training data
- **implementation**
  - **source files** – source code of the program
  - **documentation** – documentation of the source code
- **text**
  - **latex** – source files for PDF generation
  - **pdf** – Master's thesis in PDF format

Appendix B

Generator Architecture

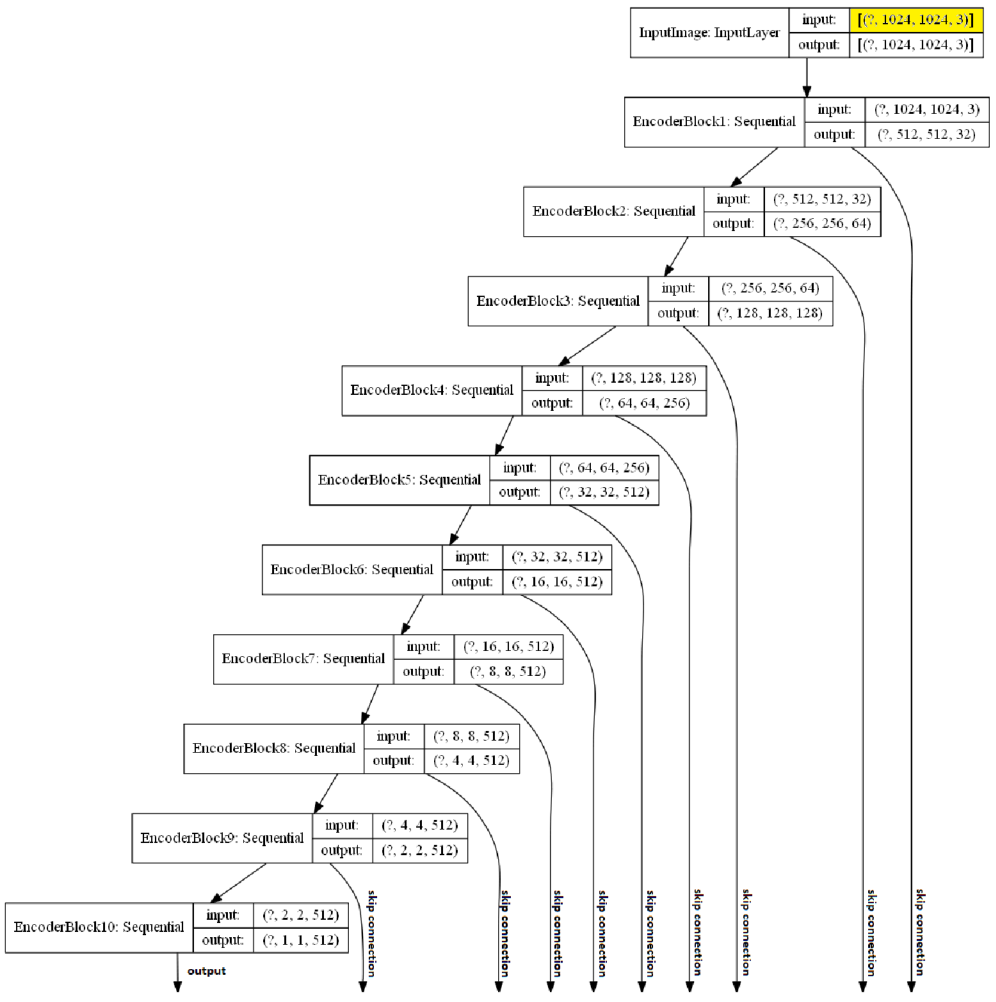


Figure B.1: Architecture of the encoder part of the SRIG generator. Outputs of the encoder are connected to the decoder inputs in Figure B.2.

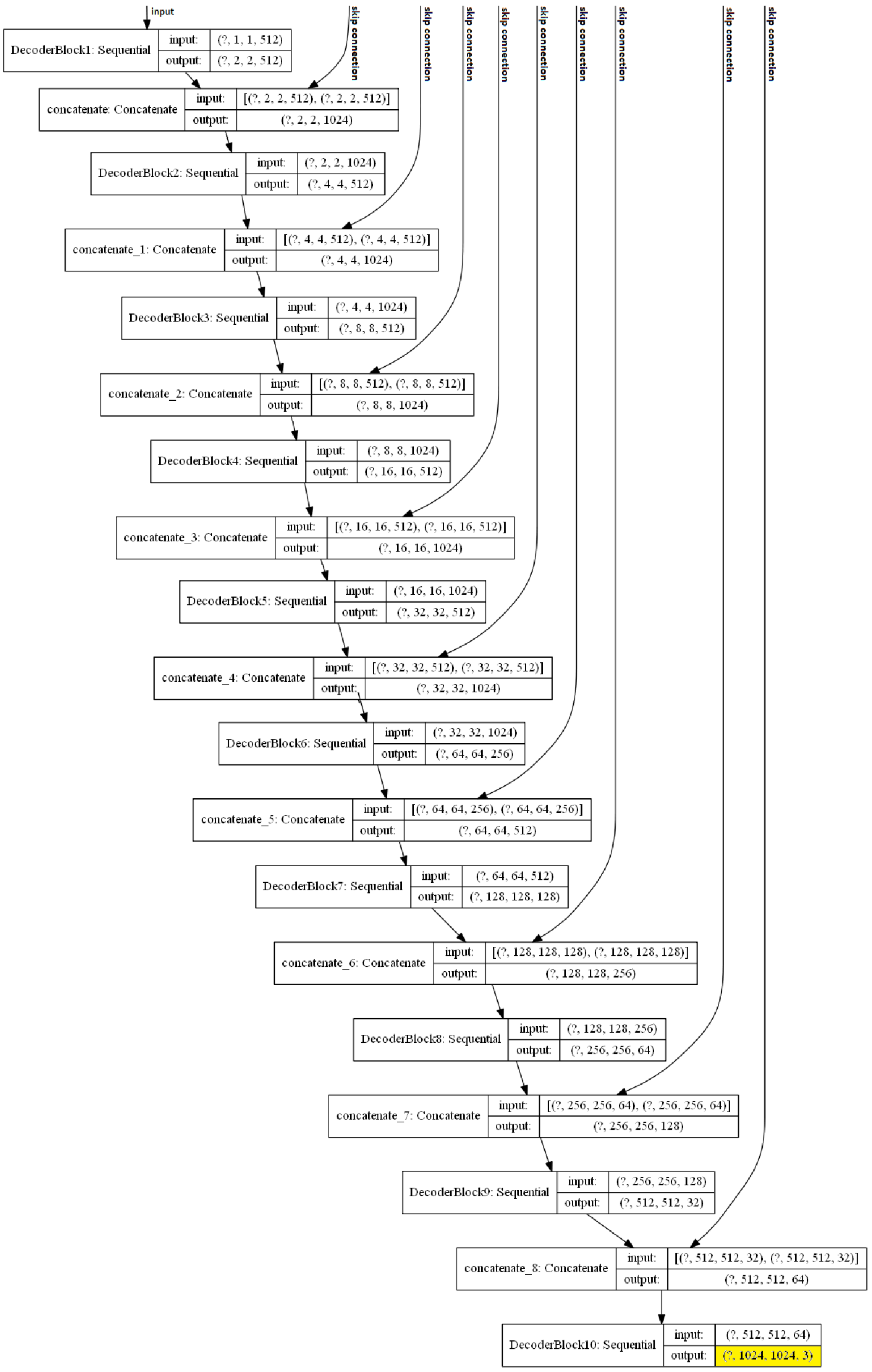

input
skip connection
skip connection
skip connection
skip connection
skip connection
skip connection
skip connection
skip connection
skip connection
DecoderBlock1: Sequential
input: (?, 1, 1, 512)
output: (?, 2, 2, 512)
concatenate: Concatenate
input: [(?, 2, 2, 512), (?, 2, 2, 512)]
output: (?, 2, 2, 1024)
DecoderBlock2: Sequential
input: (?, 2, 2, 1024)
output: (?, 4, 4, 512)
concatenate_1: Concatenate
input: [(?, 4, 4, 512), (?, 4, 4, 512)]
output: (?, 4, 4, 1024)
DecoderBlock3: Sequential
input: (?, 4, 4, 1024)
output: (?, 8, 8, 512)
concatenate_2: Concatenate
input: [(?, 8, 8, 512), (?, 8, 8, 512)]
output: (?, 8, 8, 1024)
DecoderBlock4: Sequential
input: (?, 8, 8, 1024)
output: (?, 16, 16, 512)
concatenate_3: Concatenate
input: [(?, 16, 16, 512), (?, 16, 16, 512)]
output: (?, 16, 16, 1024)
DecoderBlock5: Sequential
input: (?, 16, 16, 1024)
output: (?, 32, 32, 512)
concatenate_4: Concatenate
input: [(?, 32, 32, 512), (?, 32, 32, 512)]
output: (?, 32, 32, 1024)
DecoderBlock6: Sequential
input: (?, 32, 32, 1024)
output: (?, 64, 64, 256)
concatenate_5: Concatenate
input: [(?, 64, 64, 256), (?, 64, 64, 256)]
output: (?, 64, 64, 512)
DecoderBlock7: Sequential
input: (?, 64, 64, 512)
output: (?, 128, 128, 128)
concatenate_6: Concatenate
input: [(?, 128, 128, 128), (?, 128, 128, 128)]
output: (?, 128, 128, 256)
DecoderBlock8: Sequential
input: (?, 128, 128, 256)
output: (?, 256, 256, 64)
concatenate_7: Concatenate
input: [(?, 256, 256, 64), (?, 256, 256, 64)]
output: (?, 256, 256, 128)
DecoderBlock9: Sequential
input: (?, 256, 256, 128)
output: (?, 512, 512, 32)
concatenate_8: Concatenate
input: [(?, 512, 512, 32), (?, 512, 512, 32)]
output: (?, 512, 512, 64)
DecoderBlock10: Sequential
input: (?, 512, 512, 64)
output: (?, 1024, 1024, 3)

Figure B.2: Architecture of the decoder part of the SRIG generator. Inputs of the decoder are connected to the encoder outputs from Figure B.1.