Universitat de Girona
Escola Politècnica Superior

Treball Final de Màster

Estudi: Màster en Ciència de Dades

Títol: Una Plataforma per Classificar Melanomes

Document: Memòria

Alumne: Wilber Eduardo Bermeo Quito

Tutor: Rafael Garcia Campos
Departament: ARQUITECTURA I TECNOLOGIA DE COMPUTADORS
Àrea: ARQUITECTURA I TECNOLOGIA DE COMPUTADORS

Convocatòria (mes/any): Setembre 2022

Universitat de Girona
Escola Politècnica Superior

MASTER'S THESIS

A Platform for Classifying Melanoma

WILBER EDUARDO BERMEO QUITO
September 2022

Master in Data Science

Advisors:

DR. RAFAEL GARCIA CAMPOS
Universitat de Girona
Departament of Computer Architecture and Technology

SR. LUIS PLA LLOPIS
Accenture S.L.

Summary

Gratitudes

Per començar vull agrair molt especialment a ...

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

1.1 Problem Statement

Skin cancer, including melanoma, represents a significant public health concern worldwide. Melanoma, in particular, poses a considerable challenge due to its high mortality rate and the need for early detection for successful treatment. Early and correct diagnosis is key for ensuring patients have the best possible prognosis. Melanoma misdiagnosis accounts for more pathology and dermatology malpractice claims than any cancer other than breast cancer, as an early misdiagnosis can significantly reduce a patient's chances of survival [L. Davis 2019].

Dermoscopy, also known as dermatoscopy *Figure 1.1*, is a noninvasive technique widely utilized for the examination of cutaneous lesions. It involves the use of a handheld instrument called a dermatoscope to visualize subsurface skin structures that are typically not visible to the naked eye. The dermatoscope illuminates the skin surface and provides magnification, allowing for a detailed examination of the epidermis, the dermoepidermal junction, and the papillary dermis. By analyzing these structures, dermatologists can identify specific features and patterns associated with various skin conditions, including melanoma [Marghoob 2022].

Figure 1.1: *During the dermoscopy procedure, the dermatologist places the dermatoscope directly on the skin, making contact with the lesion of interest. Illustration by MD Anderson Cancer Center*

Worldwide, in 2020 an estimated 324,635 people were diagnosed with melanoma and an estimated 57,043 people worldwide died from melanoma the same year [Cancer.Net 2023]. The introduction of sophisticated machinery and techniques in dermoscopy procedures *Figure 1.2* seems not enough to fight against melanoma, but the developments in artificial intelligence (AI) especially in deep learning techniques, have made Computer-Aided Diagnosis (CAD)[1] a promising path towards medical automation.

Figure 1.2: *Dermoscopy Images. Illustration by ISIC Archive*

However, there are several limitations that raise doubts about the effectiveness of automated melanoma cancer classifiers and their suitability for integration into the medical system. Firstly, certain methods are constructed based on theoretical models of melanoma appearance, which may restrict their applicability to specific morphologies and fail to capture the wide range of variations seen in real-world scenarios. Secondly, AI systems utilized in these classifiers are trained to address a singular and narrow task. Unlike human dermatologists, these systems lack the ability to consider holistic patient information when formulating a final diagnosis, reflecting the concept of weak AI [Marr 2021]. Lastly, numerous methods have been trained and evaluated using high-quality image frames, which may result in instability when applied under real-time visibility conditions where image quality is often compromised. A fundamental part of machine learning is the problem of generalization, that is, how to make sure that a trained model performs well on unseen data. If the unseen data has different distribution, i.e. a domain shift exists, the problem is significantly more difficult even the smallest changes in the statistics as compared to the training data can cause a deep neural network to fail completely [K. Stacke 2019].

Addressing these limitations and developing melanoma cancer classifiers that encompass a wider range of morphologies, incorporate holistic patient information, consider relevant elements, and demonstrate robustness in real-world

[1] Computer-Aided Diagnosis (CAD) refers to the use of computer algorithms and technologies to assist healthcare professionals in the process of medical diagnosis

scenarios are crucial for improving the reliability and effectiveness of melanoma detection and diagnosis systems.

1.2 Project Objectives

The main objective of this project is to create a health care infrastructure, focused on melanoma detection using deep learning methods to train a system capable of detecting melanoma on dermoscopy images to test the ability of computer-assisted image analysis. To this end, the gradual achievements that must be accomplished are:

- Gaining a comprehensive understanding of the theory behind deep learning and its practical applications.
- Analyzing images from dermostocopy and understanding its most important features.
- Train deep learning models with different techniques based on tranfer-learning focused on the images of the melanoma ISIC Challenge [ISIC 2019].
- Developing a CAD system. The CAD system contains the already trained models with a simple web UI[2] an API[3] and finally a mechanism using Docker to virtualize this services making it ease to deploy in any based Linux System.

1.3 Personal Motivation

I envisioned this project as a unique fusion of three personal passions. Firstly, I was fueled by a deep fascination with human cognition and reasoning. Machines, in my eyes, represented a novel paradigm through which I could delve further into this captivating realm.

I am also motivated by the remarkable problem-solving capacity of data. Regardless of its structure, data holds immense potential to uncover hidden patterns, provide insights, and drive innovation. The ability to extract meaningful information from data, regardless of its form, inspires me to constantly expand my

[2]The user interface (UI) is the point of human-computer interaction and communication in a device

[3]API stands for Application Programming Interface

knowledge and skills in order to contribute to the field of data science and make a tangible difference in the world.

Last but not least, I am driven by the immense power of automation and its ability to democratize access to research knowledge. I am amazed by how automation processes can extract value and make them readily available to professionals and the public alike.

1.4 Statement of Originality

I, Wilber Eduardo Bermeo Quito, declare that the thesis titled "A platform for classifying melanoma" is an original work completed with the support and collaboration of Accenture SL and the VICOROB laboratory.

The content presented in this thesis is the outcome of my independent research efforts, guided by the knowledge and expertise acquired through my academic studies and the valuable contributions from Accenture S.L and the VICOROB laboratory.

I acknowledge the importance of academic integrity and the consequences of plagiarism. Hence, I affirm that all the information, data, results, figures, and conclusions presented in this thesis are authentic and original. Any references or sources used have been appropriately cited and referenced.

1.5 Regulatory Framework

The inclusion of legal considerations has become a significant aspect of the field of medical imaging. Privacy concerns and the potential misuse of personal information make sharing and distributing medical data particularly challenging. To address these limitations, recent research collaborations have focused on promoting the sharing of patient data through de-identification methods. However, it is crucial to thoroughly analyze the obligations related to the protection of individuals and their personal data before engaging in projects involving medical imaging.

When working with medical images, it is the utmost importance to prioritize patient privacy rights. In the context of developing a skin lesions database, it is necessary to obtain signed consent from patients for the publication of their data. For this thesis, the ISIC Archive database was utilized, this database serves

as a publicly accessible resource for teaching, research, and the development and testing of diagnostic artificial intelligence algorithms, and it resolves any concerns related to consent [ISIC 2022]. It is a large and continually expanding open-source archive of skin images.

1.6 Related Work

Melanoma Computer-Aided Diagnosis (CAD) classifiers have been a subject of extensive research and development in recent years, also there has being a lot of work in platforms capable of explain the way these models infer. Bellow there is mentioned some remarkable related work.

Identifying Melanoma Images using EfficientNet Ensemble: Winning Solution to the SIIM-ISIC Melanoma Classification Challenge

Winning solution to the SIIM-ISIC Melanoma Classification Challenge. It is an ensemble of convolutions neural network (CNN) models with different backbones and input sizes [Q. Ha 2020a].

Dermatologist-level classification of skin cancer with deep neural networks

This study demonstrated the use of a deep learning algorithm to classify skin lesions, including melanoma, with accuracy comparable to dermatologists. The deep neural network was trained on a large dataset of images and achieved high sensitivity and specificity [A. Esteva 2017].

Computerized analysis of pigmented skin lesions: A review

The goal of this paper is to give a detailed explanation and clear up any confusion about the words and phrases used in melanoma studies. And to organize and group together useful sources, making it easier to find information on a particular sub-topic when searching through the existing literature [Korotkov 2012].

CNN-Explainer

The CNN-Explainer is an interactive web-based tool that aims to provide explanations for the predictions made by convolutional neural network (CNN) models [J. Wang 2022].

1.7 Contribution to Melanoma Detection

In this section, we present the contribution to the field of melanoma detection through the development of a melanoma CAD (Computer-Aided Diagnosis) infrastructure classifier.

The thesis comprises multiple models employing various techniques, including transfer learning, data augmentation, just-in-time testing, regularization, and others. To compare these models, tools like W&B (Weight & Biases), a MLOps platform for experiment tracking, and MLXTEND, providing utilities and extensions for machine learning and data science in Python's scientific computing stack, were utilized. The experiments were conducted using the ISIC Archive dataset, consisting of benign and malignant skin lesions. Prior to analysis, the dataset underwent preprocessing to enhance quality and normalize features. Consequently, accurate classification and differentiation of melanoma lesions from benign ones were achieved.

In order to improve the distribution and usability of all models, two services were developed. These services include a user-friendly UI and an API, both of which were containerized using Docker technology. By containerizing the UI, an intuitive and interactive user experience was provided, enabling users to seamlessly interact with the melanoma CAD system. Similarly, containerizing the API streamlined request handling and prediction serving, resulting in efficient performance. This approach facilitated easy deployment across various platforms.

CHAPTER 2

State of the Art

Melanoma, a type of cancer that arises from melanin-producing cells, can be found in various parts of the body such as the skin, eyes, nerve centers, and meninges. Early diagnosis is crucial for improving the chances of curing melanoma, even though it has the highest increasing incidence rate among all skin cancer types. According to a study, timely identification of early-stage skin cancer resulted in a significant 90% reduction in mortality rates. For instance, patients diagnosed with stage I melanoma have a 10-year overall survival probability ranging from 94% to 98%, while those in stage IV have a much lower estimated 10-year overall survival rate of just 10% to 15%.

Dermoscopy is a non-surgical method used to examine the underlying layers of the skin. While it can yield good results, it requires extensive training and experience in dermatology. However, it may not provide a definitive diagnosis for melanoma, especially in its early stages. Consequently, there is a need for an automated diagnostic tool.

In a study the classification of lesions was compared between expert opinions and artificial neural networks. The computer program demonstrated a sensitivity of 95% and a specificity of 88%, while dermatologists reported similar scores of 95% sensitivity and 90% specificity. These findings indicate the potential utilization of automated systems in the field of cancer detection [Leiter 2020].

For cancer prediction, a mostly supervised learning approach is employed that makes use of algorithms for classification based on conditional decisions or probabilities. The most common algorithms or methodologies include decision trees, convolutional neural networks (CNN), support vector machine (SVM), and k-nearest neighbors (KNN). One of the most powerful systems are CNNs (Convolutional Neural Networks), despite the fact that their use may involve a loss of explainability, which is a major concern in healthcare systems.

Although this project goes beyond the mere creation of highly accurate models for classifying Melanoma, I have taken into consideration and reviewed related works that have influenced and guided my own path.

2.1 Identifying Melanoma Images Using EfficientNet Ensemble

This is the winning approach to the SIIM-ISIC Melanoma Classification Challenge [ISIC 2020]. The team not only let the competition source code available on GitHub [Q. Ha 2020b] but they also wrote a paper explaining in detail their investigation [Q. Ha 2020a].

The project is an ensemble of convolutional neural network (CNN) models. These models utilize different backbones and input sizes, primarily focusing on image data, although some also incorporate image-level and patient-level metadata. The success of the project can be attributed to several factors:

- Implementation of a stable validation scheme.
- Effective selection of the model target.
- Thoughtful optimization of the pipeline.
- Utilization of ensemble learning with highly diverse models.

The submission that won achieved an AUC (Area Under the Curve)[1] score of 0.9600 on cross-validation and 0.9490 on the private leaderboard.

From their work, I adopted various deep learning techniques. For instance, I learned about evaluating models on unbalanced multi-class data-sets like the ISIC dataset. To properly assess the training process, they utilized the AUC with one-vs-rest (OvR) metric. This project as well uses this metric to to evaluate the performance of the models.

Additionally, this thesis incorporated an early-stop mechanism into the training process. This mechanism prevents over-fitting by monitoring the model's performance on a validation set and stopping the training if the performance starts to worsen or stagnate. This was not mentioned in the related work, but I recognized its importance and included it in my implementation.

Another aspect the current thesis borrowed from this work is the approach to cleaning and mapping data-sets from different years. They trained their models with different output sizes, but the process of mapping and joining the data-sets

[1] AUC is a metric that quantifies the overall quality of a binary classification model by measuring the area under its ROC curve. It provides a single value that summarizes the model's ability to discriminate between positive and negative instances.

into a single data-set was similar to my approach, where I consistently mapped the data into eight different classes.

The pipeline of data augmentation that is implemented in the work is also reused. As they do, this thessis use Albumentations [uslaev 2020] library instead of the native alternative transforms data agumentation from Pytorch.

These inspirations and adaptations from the related work have greatly influenced the development and organization of this deep learning project.

2.2 CNN Explainer

CNN explainer is a web page created through a collaborative research effort between Georgia Tech and Oregon State [J. Wang 2022]. It leverages TensorFlow.js, a deep learning library that utilizes GPU acceleration within the web browser, to load pretrained models for visualization. The entire interactive system is implemented in JavaScript, utilizing Svelte as a framework and D3.js for visualizations. With CNN Explainer, users can upload images and obtain predictions for those images across ten different classes.

While CNN Explainer primarily aims to serve educational purposes, my project's UI focuses on providing dermoscopy images predictions and information about the models used and their outputs. The inspiration for incorporating an interactive UI for end users stemmed from CNN Explainer. Consequently, I utilized the same web framework to develop the UI for my project.

CHAPTER 3

Domain

In this chapter, we will delve into the details of melanoma cancer, its origins, reasons for its development, and strategies to minimize the risk of its occurrence. We will also explore other common types of skin cancer and their underlying causes. By the end, I hope you have a bigger view and understanding of these conditions and be equipped with knowledge.

3.1 The Skin

The skin, our body's largest organ [Cancer.Net 2020], plays a vital role in protecting us from external threats and maintaining our overall well-being. Let's explore the three main layers of the skin in more detail:

- **Epidermis**

  The epidermis is the outermost layer of the skin, serving as a protective shield against environmental factors. It consists mainly of flat, scale-like cells called keratinocytes. These cells produce a tough protein called keratin, which helps make the skin waterproof and resistant to damage. Within the epidermis, specialized cells called melanocytes produce melanin, the pigment responsible for skin color. Melanin also helps protect the skin from harmful ultraviolet (UV) radiation.

  The epidermis is made up of several layers, including the stratum corneum, the topmost layer composed of dead skin cells that are continuously shed and replaced by new cells from the lower layers. The epidermis also contains other types of cells, such as Langerhans cells, which are part of the immune system and help defend against infections.

- **Dermis**

  Beneath the epidermis lies the dermis, a complex layer that provides structural support to the skin. The dermis contains a network of blood vessels

that supply nutrients and oxygen to the skin cells. It also houses various structures, including hair follicles, sweat glands, sebaceous glands, and nerve endings. The dermis is composed of collagen and elastin fibers, which give the skin its strength, elasticity, and flexibility. These fibers allow the skin to stretch and recoil as needed. The dermis also plays a crucial role in thermoregulation by regulating blood flow to control body temperature.

- **Hypodermis (Subcutaneous Tissue)**

  The deepest layer of the skin is called the hypodermis or subcutaneous tissue. It is primarily made up of adipose tissue, which provides insulation, cushioning, and energy storage. The hypodermis helps to regulate body temperature by acting as an insulating layer, preventing heat loss. It also acts as a shock absorber, protecting the underlying structures and organs from injury. Additionally, the hypodermis serves as a connection between the skin and the underlying muscles and bones. It contains blood vessels and nerve endings that supply nutrients and sensation to the skin.

In order to visualize the structure of the skin and understand the dimensions of each layer, *Figure 3.1* provides a representation of the three main layers of the skin: the epidermis, dermis, and hypodermis.

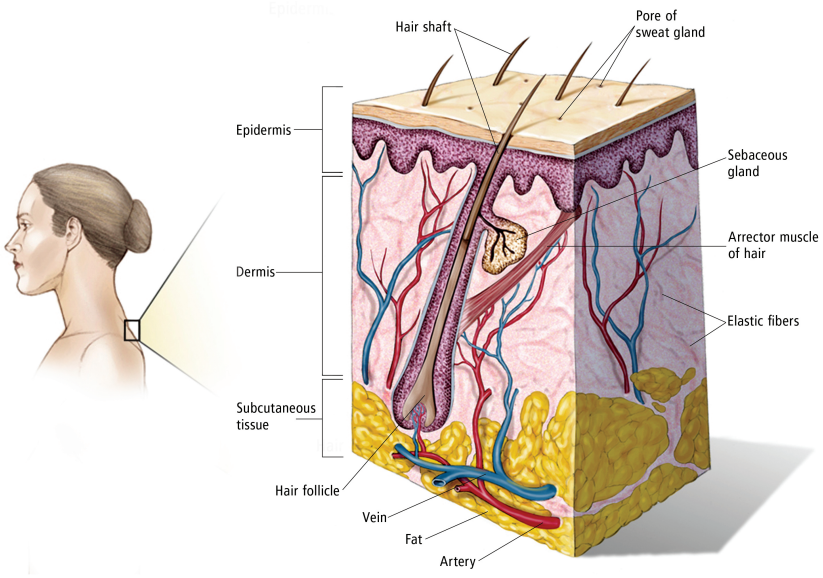


Figure 3.1: *Skin Main Layers. Illustration by Cancer.Net*

3.2 Skin Cancer

Cancer begins when healthy cells undergo changes that cause them to grow and divide uncontrollably, forming a mass known as a tumor. Tumors can be classified as either cancerous or benign. A cancerous tumor is considered malignant, meaning it has the potential to invade nearby tissues and spread to other parts of the body through a process called metastasis. Conversely, a benign tumor may grow locally but does not have the ability to spread to other areas.

Skin cancer is one of the most prevalent types of cancer, with over 3 million Americans diagnosed each year. However, the prognosis for skin cancer is generally favorable when detected early. Treatment options for skin cancer often involve topical medications, in-office procedures performed by dermatologists, or outpatient surgeries. Dermatologists specialize in diagnosing and treating conditions of the skin. As a result, skin cancer accounts for less than 1% of all cancer-related deaths.

In more advanced cases, skin cancer may require comprehensive medical care provided by a multidisciplinary team, which typically includes a dermatologist, surgical oncologist, radiation oncologist, and clinical oncologist.

3.2.1 Types of Skin Cancer

There are three primary forms of skin cancer [Cancer.Net 2020].

Basal cell carcinoma

This skin cancer arises from basal cells located in the lower epidermis. Approximately 80% to 90% of skin cancers originate from these cells, leading to the designation of basal cell carcinomas. They typically develop on the head and neck, although they can occur anywhere on the skin. Sun exposure is the primary cause, although they may also occur in individuals who underwent radiation therapy during childhood. This type of skin cancer generally grows slowly and rarely metastasizes to other parts of the body.

Squamous cell carcinoma

This skin cancer originates from flat, scale-like cells known as squamous cells that comprise a significant portion of the epidermis. Around 10% to 20% of skin

cancers develop from these cells, resulting in the classification of squamous cell carcinomas. Sun exposure is the main cause, and they can be diagnosed on various regions of the skin. They may also emerge on skin that has been burned, damaged by chemicals, or exposed to x-rays. Common sites for squamous cell carcinoma include the lips, areas with long-standing scars, and the skin surrounding the mouth, anus, and vagina. Roughly 2% to 5% of squamous cell carcinomas metastasize to other parts of the body.

Melanoma

The most aggressive type of skin cancer, originates in scattered cells known as melanocytes where the epidermis and dermis meet. Melanocytes produce the pigment melanin, responsible for skin color. Melanoma accounts for approximately 1% of all skin cancers.

3.2.2 Stages of Skin Cancer

The following stages are used for basal cell carcinoma. Basal cell carcinoma accounts for more than 90% of all skin cancers in the United States and is the most common of all cancers. Typically, it is a slow-growing cancer that seldom spreads to other parts of the body [Farber 2020].

Stage I

In stage I, abnormal cells are found in the squamous cell or basal cell layer of the epidermis. These abnormal cells may become cancer and spread into nearby normal tissue.

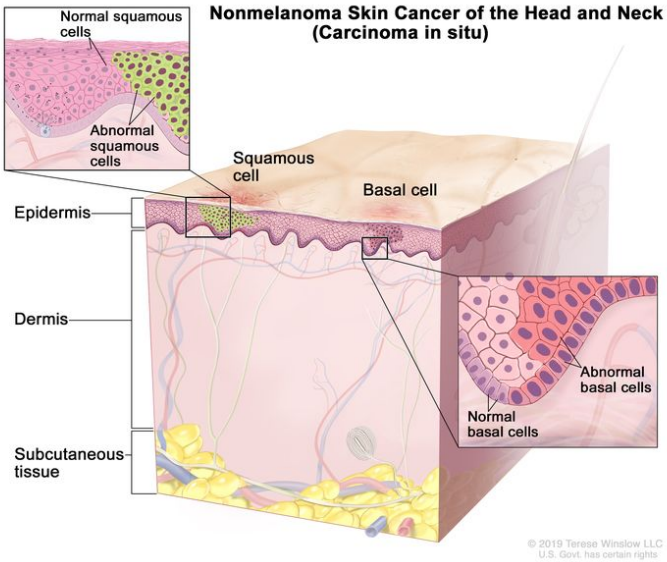


Figure 3.2: *Skin Cancer, Stage I. Abnormal cells are found in the squamous cell or basal cell layer of the epidermis. These abnormal cells may become cancer and spread into nearby normal tissue. Illustration by Terese Winslow*

Stage II

In stage II, cancer has formed and the tumor is 2 centimeters or smaller.

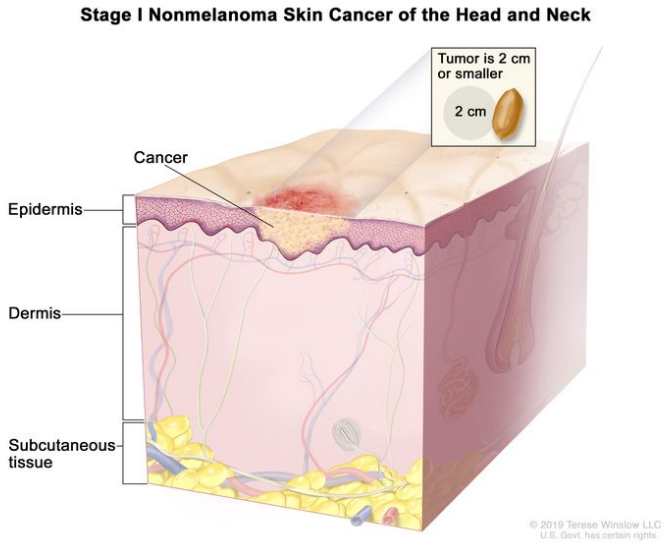


Figure 3.3: *Skin Cancer, Stage II. In stage II, cancer has formed and the tumor is 2 centimeters or smaller. Illustration by Terese Winslow*

Stage III

In stage II, the tumor is larger than 2 centimeters but not larger than 4 centimeters.

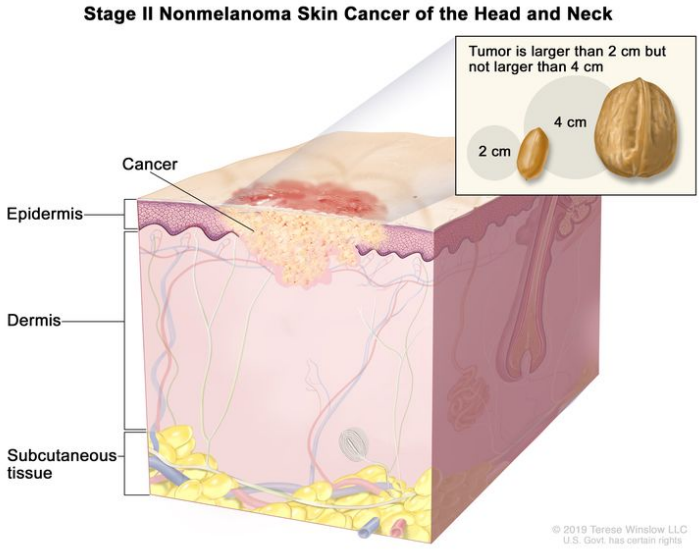


Figure 3.4: *Skin Cancer, Stage III. The tumor is larger than 2 centimeters but not larger than 4 centimeters. Illustration by Terese Winslow*

Stage IV

In this stage, the cancer may spread to the rest of the body covering the nerves bellow the dermis or bellow the subcutaneous tissue or the bones.

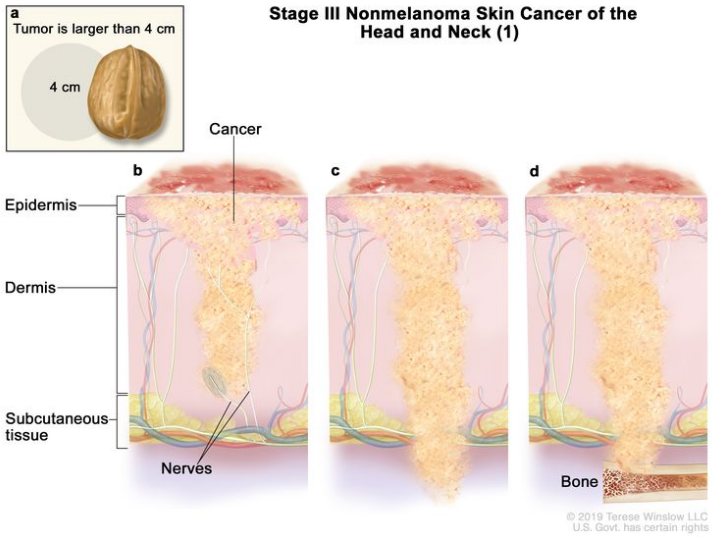


Figure 3.5: *Skin Cancer, Stage IV. The tumor is larger than 2-4 centimeters and start spreading. Illustration by Terese Winslow*

3.2.3 Associated Factors and Preventions of Skin Cancer

The risk factors associated with skin cancer encompass sun exposure, fair skin, as well as certain physical characteristics such as blond hair and blue eyes. The presence of melanin, a protein responsible for the skin's color or pigment, plays a crucial role in shielding the skin from ultraviolet (UV) radiation. Consequently, individuals with lighter skin or lower levels of melanin possess reduced UV protection [Partridge 2013].

Although skin cancer may initially appear to predominantly affect individuals with light complexions, those with dark skin are also susceptible. They may observe signs of cancer on the palms of their hands or soles of their feet.

Advancements in research have led to treatments targeting the genetic level. New drug therapies assist in stimulating the immune system to produce antibodies capable of combatting rapidly dividing cells. While these therapies may have potential side effects, they are generally milder than those associated with chemotherapy. Moreover, they result in an enhanced immune system that is better equipped to battle cancer.

A patient's overall health also appears to influence their ability to fight cancer, although the precise connection between one's health and their risk of developing cancer remains unclear. However, it can be assumed that better overall health enhances the immune system's capacity to combat cancer.

Prevention of skin cancer is straightforward: minimizing exposure to the sun and UV radiation. Ceasing the use of tanning beds is advised. When exposed to the sun, wearing sunscreen and a wide-brimmed hat that provides comprehensive head shading is recommended [Partridge 2013].

Consideration should also be given to wearing clothing with SPF protection. Some clothing manufacturers now produce summer attire with SPF levels similar to those found in sunscreen. This is significant since most regular t-shirts offer only minimal SPF protection of around 2.

Skin cancer ranks among the most prevalent forms of cancer, yet it is also one of the most easily detectable and treatable. Taking proactive measures to prevent sun exposure is crucial, while early detection is highly important. The prognosis is favorable when melanoma is identified in its early stages.

CHAPTER 4

Preliminaries

The objective of this chapter is to provide a comprehensive overview that defines the scope of the thesis. It covers essential theoretical concepts necessary to comprehend the experiments presented in subsequent chapters, delves into the technical knowledge required to understand the functioning of the CAD infrastructure, and explores the study of the dataset used for conducting the thesis.

4.1 Theoretical Background

This section provides an introduction to the theoretical foundations of training deep learning models. We will cover essential concepts such as neural networks, activation functions, loss functions, gradient descent. Understanding these concepts is crucial for comprehending the inner workings of deep learning algorithms and to know if the trained thesis models are performing correctly.

4.1.1 Artificial Intelligence

AI is one of the newest fields in science and engineering. Work started in earnest soon after World War II, and the name itself was coined in 1956 attributed by John McCarthy of MIT.

After successfully decrypting Enigma[1], in World War II, the mathematician Alan Turing posed a groundbreaking question that had a profound impact on society: "Can machines think?" [Turing 1950]. In this seminal work, Turing not only outlined the process of developing intelligent machines but also presented a method for assessing their intelligence. Even today, the Turing Test remains a widely recognized benchmark for evaluating the intelligence of artificial systems. According to this test, if a human cannot distinguish, through a text-based conversation, whether they are interacting with a human or a machine, the artificial system can be considered to possess intelligence.

[1] Enigma was a rotor machine designed to encrypt and decrypt messages.

The term "AI" is exciting but, the definition of this term is blurry [Russell 1950]. Historically there are two approaches to *AI* have been followed, each by different people with different methods. The human-centered approach and the rationalist approach. The incorporation of a human-centered approach into the field must incorporate empirical scientific methods, which entail making observations and formulating hypotheses about human behavior. On the other hand, a rationalist approach combines the disciplines of mathematics and engineering. These different perspectives have both faced criticism and received praises.

From that juncture until the turn of the century, the field of *AI* encountered fluctuations, witnessing periods of remarkable accomplishments interspersed with the infamous *AI* winters. Currently *AI* is a hot topic of discussion, evident from the increasing searches and online mentions. While *AI* has been around for more than 65 years, the recent exponential trend can be attributed to the availability of cost-efficient storage, transfer, and computation capabilities for massive amounts of data. This newfound capacity was previously unimaginable and has contributed to the current *AI* boom.

AI has expanded into numerous distinct fields, showcasing its extensive range of applications, techniques, and interdisciplinary nature. The diversity of branches in the field of artificial intelligence is represented in *Figure 4.1*, which depicts several current *AI* fields. However, it is important to note that the figure can be further expanded to include additional fields, such as Speech Processing and Expert Systems, among others.

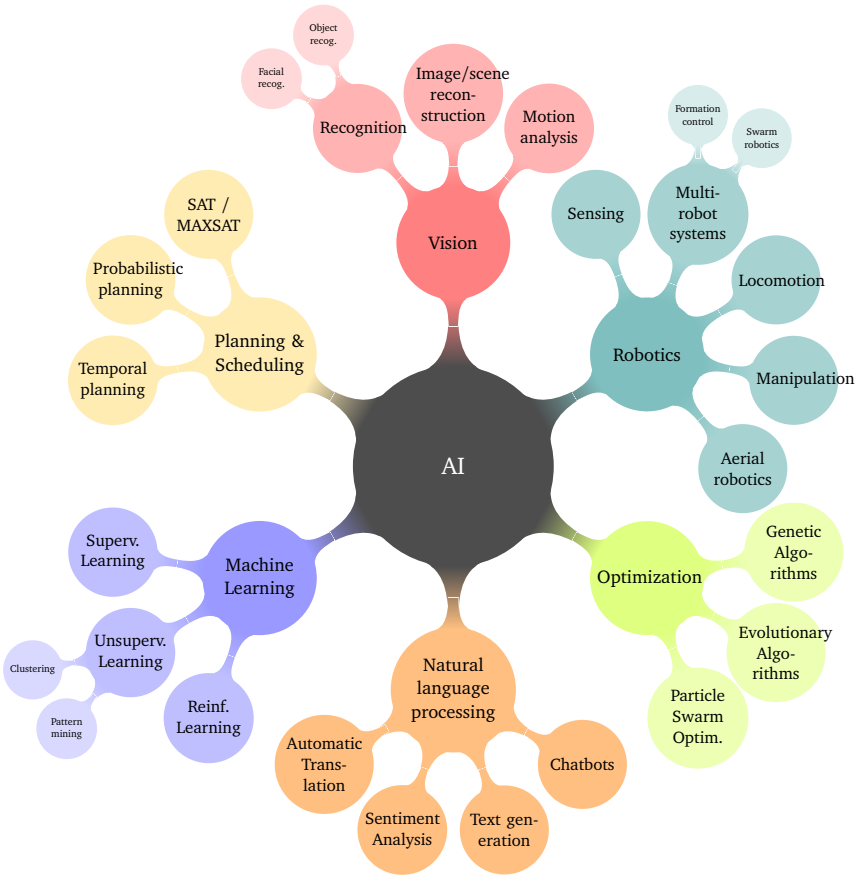


Figure 4.1: *AI Sub-Specialists. Note that these subfields often intersect and combine. Illustration by Author*

4.1.2 Machine Learning

Machine learning is a branch of artificial intelligence (*AI*) and computer science which focuses on the use of data and algorithms to imitate the way that humans learn, gradually improving its accuracy [IBM 2020]. The term *AI* was defined by the pioneer Arthur Samuel as "the field of study that gives computers the ability to learn without explicitly being programmed."

The central idea of machine learning is the existence of a mathematical relationship between any combination of input and output data. Rather than manually programming knowledge into computers, machine learning endeavors to automatically learn significant relationships and patterns by observing examples.

There are three subcategories of machine learning [IBM 2021]. These subcategories are listed bellow.

Supervised Learning

Machine learning models are trained with labeled data sets, which allow the models to learn and grow more accurate over time. For example, an algorithm would be trained with pictures of dogs and other things, all labeled by humans, and the machine would learn ways to identify pictures of dogs on its own. Supervised machine learning is the most common type used today.

Unsupervised Learning

Unsupervised learning involves using a training set that comprises unlabeled inputs, meaning inputs that do not have any assigned desired output. The primary objective of this learning approach is to uncover concealed patterns or data clusters without relying on human intervention.

Reinforcement Learning

Reinforcement learning occupies a position between supervised and unsupervised learning. In a certain sense, there is some form of supervision, but it does not involve explicitly specifying a desired output for every input in the data. Instead, a reinforcement learning algorithm receives feedback from the environment only after selecting an output, known as an action, based on a given

input or observation. The feedback indicates the extent to which the action fulfills the learner's goals.

Figure 4.2 is graphical representation of the subcategories of machine learning.

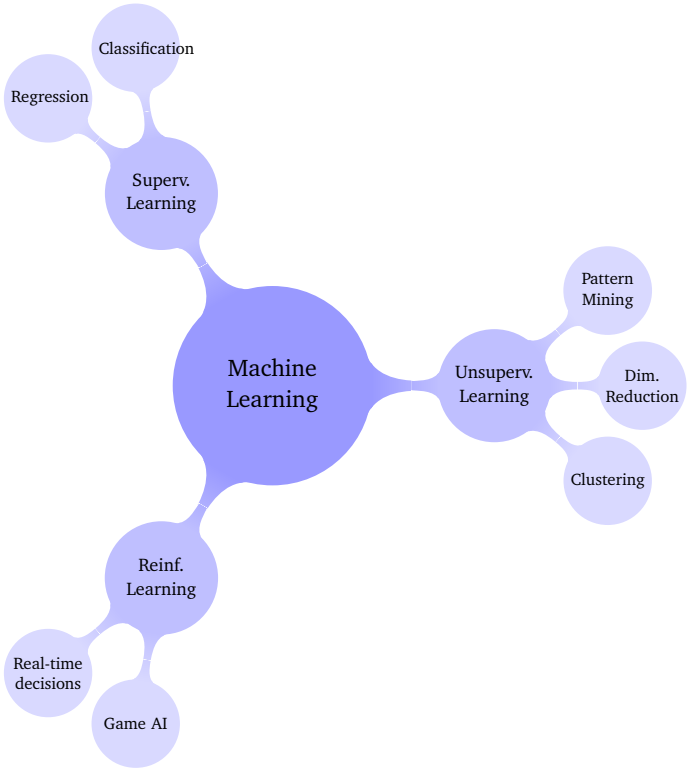


Figure 4.2: *Subcategories of Machine Learning. Illustration by Author*

4.1.3 Deep Learning

Deep learning, is a subset of machine learning algorithms, is primarily associated with supervised learning techniques. This algorithms derive conclusions by analyzing data with a given logical structure. In contrast to traditional machine learning algorithms, deep learning algorithms operate at a higher level of abstraction. They possess the ability to perform the laborious process of feature extraction. Each algorithm applies a nonlinear transformation to its input and uses the acquired knowledge to generate a statistical model as the output. This iterative process continues until the output reaches an acceptable level of accuracy. The term "deep" refers to the number of layers the data must pass through during these iterations.

Deep learning has been successfully applied to various problems, ranging from

self-driving cars to speech recognition. In this particular project, deep learning is employed to classify whether a given dermoscopy image is a melanoma or not. To achieve this, it has been necessary to understand how convolutional neural networks work, address certain issues that affect the precision of these models, explain the key metrics primarily used to measure the accuracy of these models.

4.1.4 Deep Neural Networks

Deep neural networks are part of the deep learning algorithms within the realm of machine learning, which, as mentioned earlier, falls under the umbrella of artificial intelligence. *Figure 4.3* is an illustration of the deep learning family.

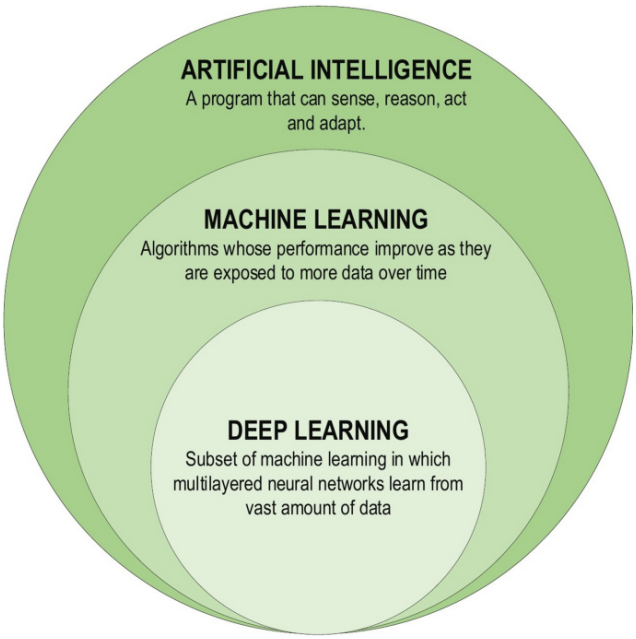


Figure 4.3: *Deep Learning Family. Illustration by WIKIPEDIA*

Deep neural networks, also known as fully connected layers, are modeled after the human brain. These algorithms achieve their functionality by propagating signals from the input layer to the output layer while passing through intermediate hidden layers. Each layer consists of neurons responsible for detecting patterns from the incoming connections. These neurons perform a crucial task by combining input from the data with a corresponding set of coefficients, also known as weights. These weights act as amplifiers or dampeners, modulating the impact of the input on the neuron's activation.

By leveraging this mechanism, each layer in the deep neural network acquires the capability to capture distinctive features from the input data. Notably, as the signal travels deeper into the network, the extracted features become increasingly sophisticated and abstract. This characteristic arises from the hierarchical nature of the network architecture, enabling the model to learn and discern intricate patterns and representations.

The process of feature extraction in deep neural networks allows the network to progressively learn and uncover complex relationships within the data. As the signal traverses through the network's layers, each subsequent layer builds upon the extracted features from the previous layer, resulting in the generation of more intricate and comprehensive representations. This ability to capture and model hierarchical features is a significant advantage of deep neural networks, making them particularly effective in tasks requiring the recognition of complex patterns and high-level abstractions.

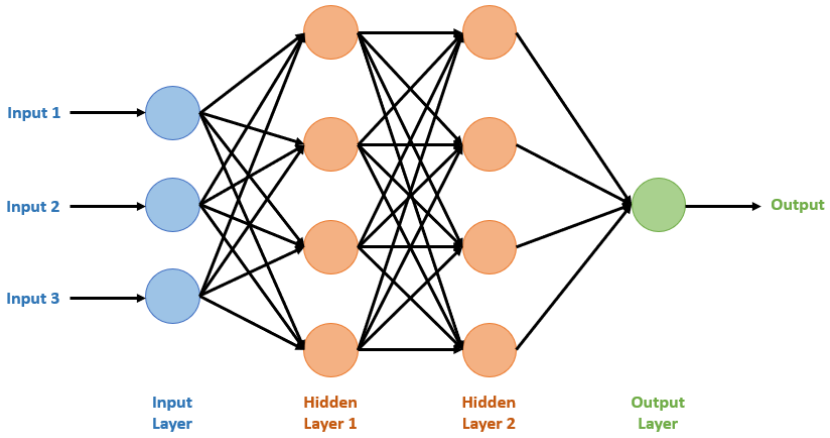


Figure 4.4: *Deep Neural Network. This NN has two hidden layers. Illustration by Introduction to Neural Networks in Deep Learning*

The minimum unit in a layer is known as neuron or perceptron. The perceptron, is the oldest simple neural network, created by Frank Rosenblatt in 1958. Every node within the network possesses an associated value, which is computed based on the incoming nodes and edges, as depicted in *Figure 4.5*. The computation takes place in the following manner: the products of inputs and their respective weights are summed with a bias, and this sum is subsequently passed through the node's activation function. The activation function serves the purpose of compressing the resulting value within the range of 0 to 1 and introduces non-linearity, allowing the model to learn complex relationships in the data. The resulting value determines the degree to which the signal should

propagate through the network, ultimately influencing the final outcome.

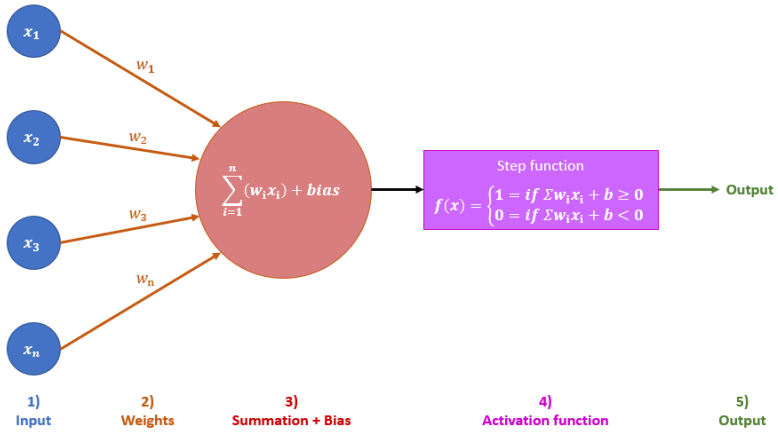


Figure 4.5: *The Perceptron. Illustration by Introduction to Neural Networks in Deep Learning*

4.1.5 Convolutional Neural Networks

A convolutional neural network (CNN) is a specific type of neural network that is particularly well-suited for analyzing visual imagery. It draws inspiration from the organization of the animal visual cortex, which is responsible for visual processing in living organisms.

One of the key innovations of CNNs is their ability to automatically learn a large number of filters in parallel. These filters are learned during the training process and are specifically tailored to solve a particular predictive modeling problem, such as image classification. By learning these filters, CNNs become adept at extracting relevant features and patterns from visual data without the need for manual feature engineering.

CNNs are designed to learn spatial hierarchies of features through a process called backpropagation. During training, the network adapts its internal parameters, or weights, by iteratively propagating the error backwards from the output to the input layers. This allows the network to gradually adjust its weights in a way that minimizes the difference between its predicted outputs and the true outputs.

One key distinction of CNNs compared to regular neural networks is their use of parameter sharing. In a CNN, all neurons within a particular feature map share the same weights. This sharing of weights significantly reduces the number of

parameters in the network, making it more computationally efficient. By sharing weights, the network can detect the same patterns or features regardless of their spatial position in the input data. This property, known as translation invariance, enables CNNs to recognize objects or patterns regardless of their location within an image.

Furthermore, CNNs have a three-dimensional arrangement of neurons: width, height, and depth. The width and height dimensions correspond to the spatial dimensions of the input data, such as the width and height of an image. The depth dimension refers to the number of channels or feature maps in each layer, where each channel represents a different aspect or feature of the input.

To build CNN architectures, there are four main types of layers used, and they are listed bellow.

Convolutional Layers

A convolutional layer is responsible for detecting and extracting features from the input data. It consists of a set of learnable filters, also known as kernels or feature detectors. Each filter is a small matrix of weights that slides or convolves across the input data to perform a dot product operation at each spatial location.

The filter slides or convolves across the input data with a defined stride, which specifies the amount of shift between each position. At each spatial location, a dot product is computed between the filter and the overlapping region of the input. The dot product involves element-wise multiplication of the filter values with the corresponding input values, followed by summing up the results.

The result of each dot product operation is a single value, forming a pixel in the output feature map. Additionally, an activation function is applied element-wise to introduce non-linearity into the network.

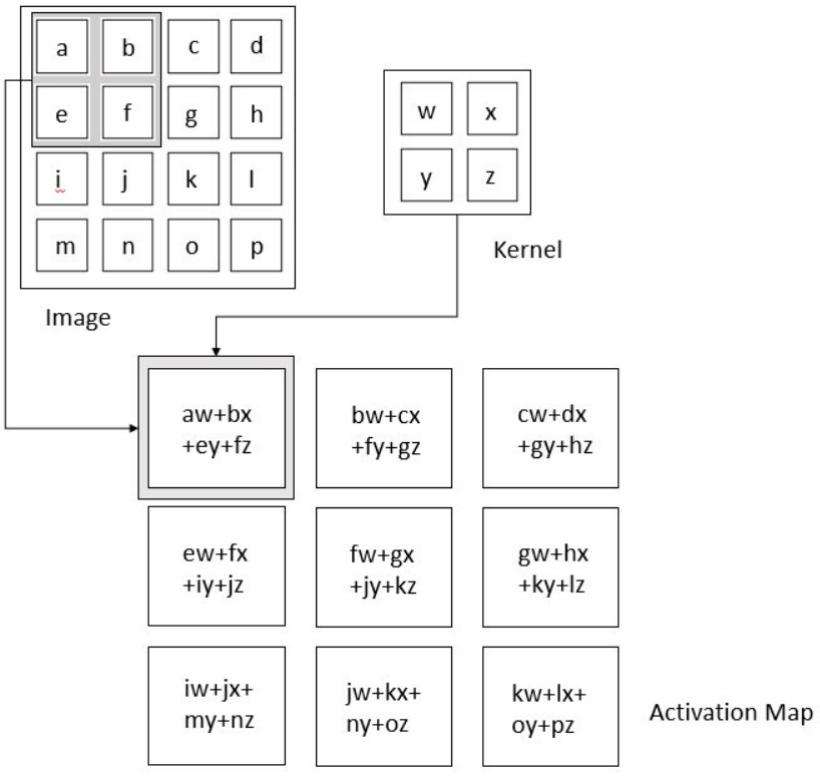


Figure 4.6: *Convolutional Operation on Input Image. Illustration by towardsdata-science*

Pooling Layers

Pooling layers performs a summarizing of nearby outputs in the network, effectively replacing certain locations with a condensed representation. This reduces the spatial dimensionality of the output, leading to decreased computational requirements and weight parameters. The pooling operation is applied independently to each slice of the representation.

Various pooling functions exist, including averaging the values within a rectangular neighborhood, computing the L2 norm of the neighborhood, or using a weighted average based on distance. However, the most widely used method is max pooling, which selects the maximum output value from the neighborhood.

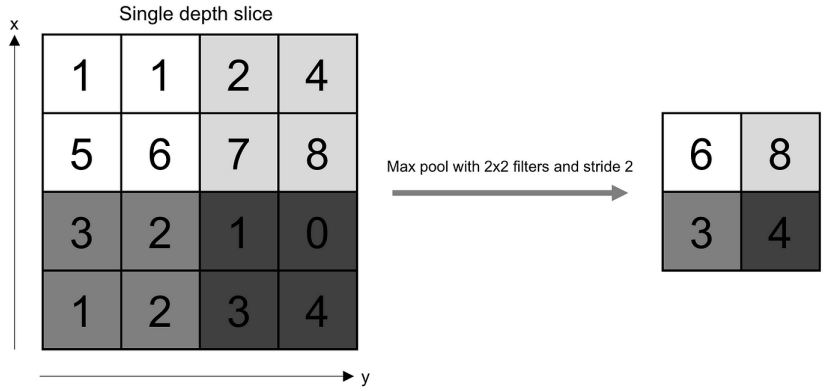


Figure 4.7: *Polling Operation on Input Image. Illustration by towardsdatascience*

Fully Connected Layers

Neurons in this layer have full connectivity with all neurons in the preceding and succeeding layer. This is why it can be computed as usual by a matrix multiplication followed by a bias effect. The topology in the fully connected portion of a CNN depends on various factors, including the complexity of the task, the nature of the data, and the capacity of the model. There is no fixed rule for determining the exact number of neurons or layers, and it often requires experimentation and tuning.

The fully connected layer helps to map the representation between the input and the output. See *Figure 4.4* to a visual representation of a fully connected layer.

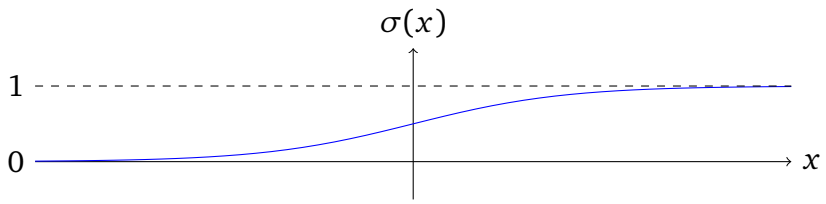
Non-linearity Layers

Convolutional operations in neural networks are linear, meaning they capture linear patterns in the data. However, many objects in images exhibit non-linear characteristics that cannot be effectively detected using linear operations alone. To address this limitation, non-linearity layers, such as activation functions, are typically applied immediately after the convolutional layer. These non-linearity layers introduce non-linear transformations to the activation maps, enabling the network to learn and detect complex, non-linear patterns in the data.

There are many types of non-linearity functions applied after the convolutional layers, the most popular are:

- **Sigmoid**

  The sigmoid function takes a real-valued number and "squashes" it into a range between 0 and 1.

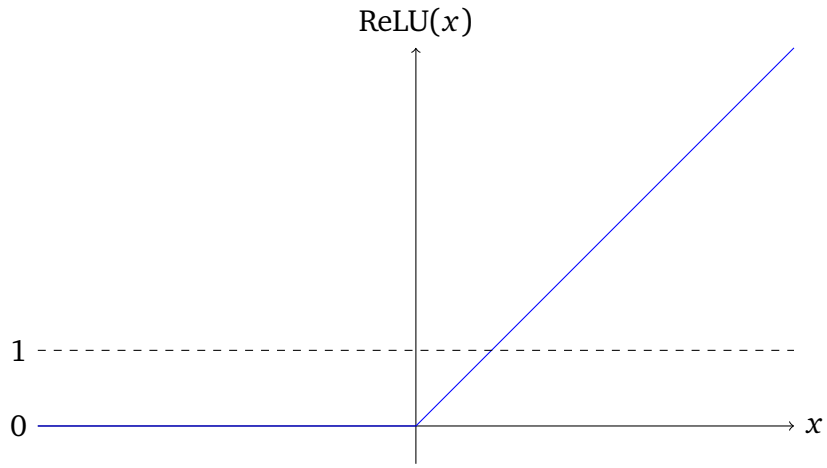


  Mathematically, the sigmoid function is expressed as:

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

- **ReLU**

  The Rectified Linear Unit (ReLU) is maybe the most popular activation functions in the last few years. The function is just simply threshold at zero.

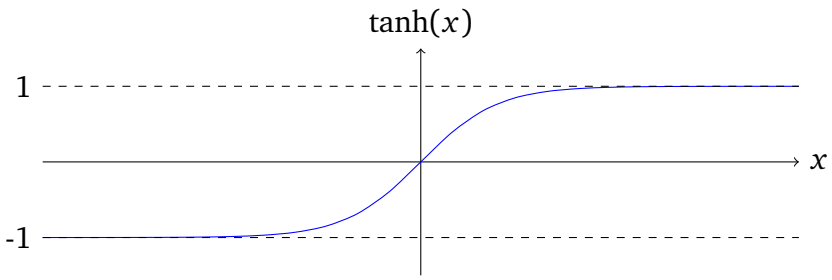


Mathematically, the ReLU function is expressed as:

$$f(x) = \max(0, x)$$

- **Tanh**

  The hyperbolic tangent function tanh compresses a real-valued number to the range of [-1, 1]. Similar to the sigmoid function, tanh exhibits saturation, but unlike sigmoid, its output is centered around zero.



Mathematically, the tanh function is expressed as:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

4.1.6 Loss Function

A loss function, also known as a cost function or an objective function, is a measure used in machine learning and optimization algorithms to quantify how well a model performs on a given task. It represents the discrepancy between the predicted outputs of a model and the true values or labels associated with the training data.

Different machine learning tasks and algorithms may use different loss functions, depending on the specific problem being addressed. Commonly used loss functions include:

- Mean Squared Error (MSE)
- Binary Cross-Entropy
- Mean Absolute Error (MAE)
- Cross-entropy Loss

For the thesis, the loss function used to train the models is Cross-entropy Loss as this a multiclass problem. Please, don't confuse the optimizer algorithm with the metric used to evaluate how good are the models. To evaluate how good are the models I used the ROC AUC with one-vs-rest strategy, explained in *Section 4.1.7*.

Cross-entropy loss, also known as log loss, is a widely used loss function in machine learning, particularly in classification tasks. It measures the dissimilarity between predicted class probabilities and true class labels.

In multi-class classification, the cross-entropy loss is calculated using the true class labels y and the predicted class probabilities p for each class. It can be defined as:

$$-\sum_{i=1}^{C} y_i \log(p_i)$$

where:

- C is the total number of classes.
- y_i represents the true label for class i.
- p_i represents the predicted probability for class i.

4.1.7 Metrics

There are a considerable amount of different metrics that can be computed for a model prediction but I'll present the essential metrics used in the thesis.

Confusion Matrix

A confusion matrix is a square matrix with dimensions `NxN`, where `N` represents the total number of classes being predicted. It provides a visual representation of the errors or confusion made by a classification model during its predictions. The matrix, illustrated as *Figure 4.8*, is structured with columns and rows representing the predicted and actual classes, respectively.

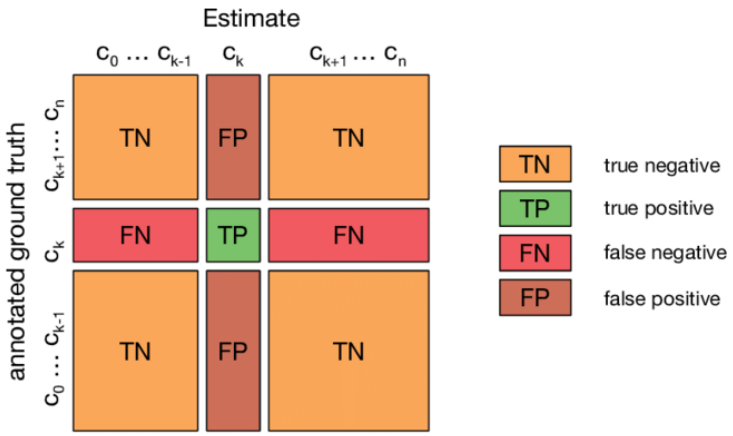


Figure 4.8: *Confusion Matrix Multi-Class. Illustration by WIKIPEDIA*

- **True Positive (TP)**

  This refers to an outcome where the model accurately predicts the positive class.

- **False Positive (FP)**

  This describes a situation where the model mistakenly predicts the positive class when it should have predicted the negative class.

- **True Negative (TN)**

  This represents an outcome where the model correctly predicts the negative class.

- **False Negative (FN)**

This indicates a scenario where the model incorrectly predicts the negative class instead of predicting the positive class.

In a logical sense, when we add up the elements on the main diagonal of the confusion matrix, we obtain the total number of correct predictions. Conversely, summing the elements on the antidiagonal provides us with the total number of incorrect predictions. Utilizing these values, we can compute additional metrics such as accuracy, precision, recall, specificity, and F1 score.

Accuracy

Accuracy is a commonly employed and straightforward performance metric in classification tasks. It calculates the ratio of correct predictions to the total number of predictions made on a dataset, thus determining the likelihood of correctly classifying an input. Accuracy is especially valuable when working with a balanced dataset, where the number of instances in each class is roughly equivalent. However, in the case of the previously mentioned thesis dataset, which is not balanced, the use of accuracy serves primarily to monitor model training performance rather than being the main evaluation metric.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

TPR

TPR stands for True Positive Rate, and it is also known as sensitivity or recall. It measures the proportion of actual positive cases that are correctly identified as positive by a classification model or test.

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

FPR

FPR stands for False Positive Rate, and it measures the proportion of actual negative cases that are incorrectly identified as positive by a classification model or test.

$$\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}}$$

AUC-ROC Curve with One-vs-Rest Strategy

The ROC Curve, when computed with the one-vs-rest (OvR) strategy for multi-class classification, provides insights into how well the model can differentiate a class from the rest of the classes. It is a probabilistic curve that plots the true positive rate (TPR) against the false positive rate (FPR). By treating the target class as the positive class and the remaining classes as the negative class in separate binary classification tasks (*Figure 4.9*).

The area under the curve (AUC) is a value between 0 and 1 that measures the ability of a classifier to distinguish between classes. It is used as a summary of the ROC curve. The higher the AUC, the better the performance of the model at distinguishing between the positive and negative classes.

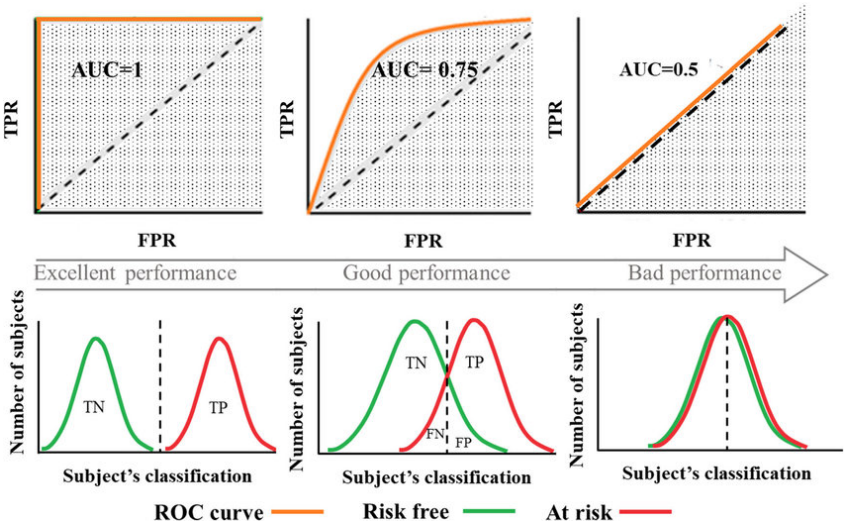


Figure 4.9: *AUC-ROC. Comparison of multiple ROC curves and how much overlap there are between classes. Illustration by Elizabeth Louise Thomas*

4.1.8 Optimizer

An optimizer is an algorithm that finds the value of the parameters (weights) that minimize the error when mapping inputs to outputs. These optimization algorithms widely affect the accuracy and speed training of the deep learning models.

While training a deep learning models, the optimizer modifies the weights of the model in each epoch to minimize the loss function.

The are a considerable amount of different optimizer algorithms out there with their pros and cons. In this section I present the family of optimizers based on Gradient Descent optimization. This family of optimizer algorithms were used for the thesis.

Gradient Descent

Using the Gradient Decent optimization algorithm, the weights are updated incrementally after each epoch (pass over the training dataset).

The magnitude and direction of the weight update is computed by taking a step in the opposite direction of the cost gradient.

$$\nabla J(w) = (\frac{\partial J}{\partial w_1}, \frac{\partial J}{\partial w_2}, \ldots, \frac{\partial J}{\partial w_n})$$

$$\Delta w = -\eta \cdot \nabla J(w)$$

Finally, the weight are updated after each epoch with the following expression:

$$w = w + \Delta w$$

- Δw represents the weight update.
- η (eta) denotes the learning rate, which controls the step size of the update.
- $\frac{\partial J}{\partial w_i}$ represents the partial derivative of the cost function J with respect to the weight w_i.
- $\nabla J(w)$ represents the gradient of the cost function $J(w)$ with respect to the weights w.

In Gradient Descent optimization, we compute the cost gradient based on the complete training set; hence, we sometimes also call it batch gradient descent. In case of very large datasets, using Gradient Descent can be quite costly since we are only taking a single step for one pass over the training set – thus, the larger the training set, the slower our algorithm updates the weights and the longer it may take until it converges to the global cost minimum.

The next lines are a high level pseudo-implementation of the Gradient Descent algorithm:

- for each epoch:
  - for each weight j:
    - $w_j = w_{j-1} + \Delta w_j$

The *Figure 4.10* shows how the Gradient Descent optimizer would try to modify the the weight values that minimizes the cost function.

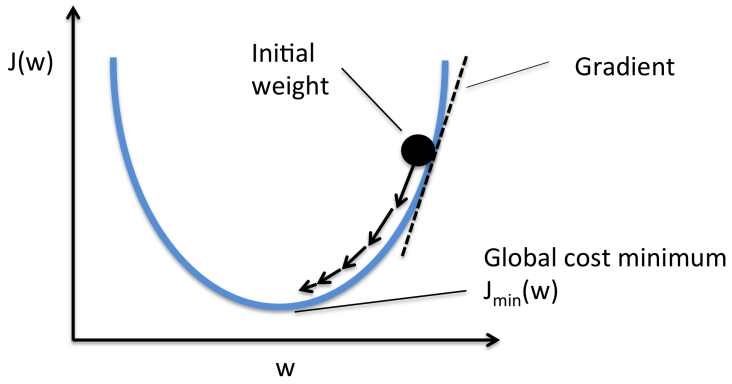


Figure 4.10: *Function Optimization. Optimization of a 2D function using Gradient Descent algorithm. Illustration by sebastianraschka*

Stochastic Gradient Descent

Stochastic Gradient Descent work similar to the Gradient Descent but the weight updates are not accumulated as we've seen above for Gradient Descent. Instead, the weights are updated after each training sample. Due to its stochastic nature, the path towards the global cost minimum is not "direct" as in Gradient Descent, but may go "zig-zag"if we are visuallizing the cost surface in a 2D space, see *Figure 4.11*. However, it has been shown that Stochastic Gradient Descent almost surely converges to the global cost minimum if the cost function is convex.

The next lines are a high level pseudo-implementation of the Stochastic Gradient Descent algorithm:

- for each epoch or until approx. cost minimum is reached:
  - for training sample i:
    - for each weight j:
      - $w_j = w_{j-1} + \Delta w_j$, where $\Delta w_j = \eta(target^{(i)} - output^{(i)})x_j^{(i)}$

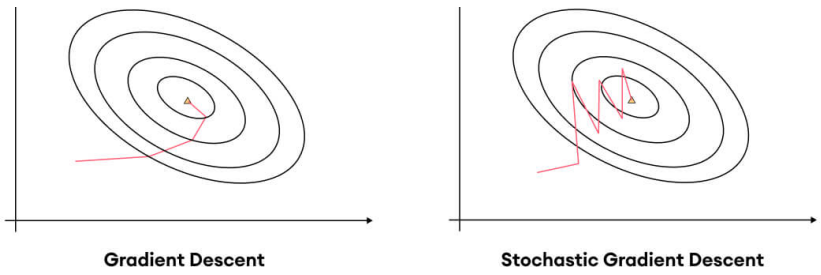


Figure 4.11: *Gradient Descent vs Stochastic Gradient Descent. It illustrates the optimization process of both algorithms. Illustration by superannotate*

4.1.9 Forward Propagation and Backward Propagation

Forward and backpropagation are fundamental processes in training neural networks and optimizing their parameters. They play a crucial role in enabling the network to learn from data and improve its performance over time.

Resuming the forward-propagation and back-propagation (*Figure 4.12*), in order to find the direction of the steepest descent (minimising the overall loss function), we need to calculate gradients of the loss function with respect to weights and bias. After that, we'll be able to update weights and bias using negative gradients multiplied by the learning rate.

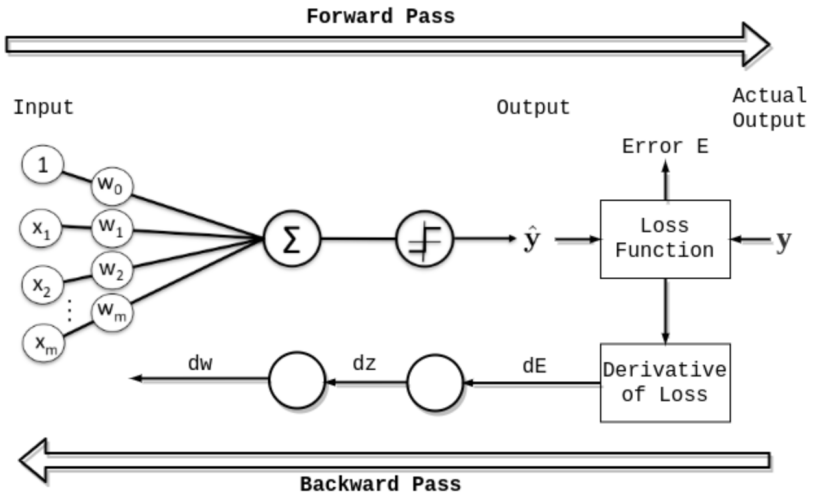


Figure 4.12: *Forward Propagation and Backward Propagation. Illustration by baeldung*

Forward Propagation

Forward propagation refers to the process of computing the output of a neural network given an input or a batch of inputs. During forward propagation, the input data flows through the network's layers in a sequential manner, from the input layer to the output layer. Each layer performs a series of computations, typically involving linear transformations (such as matrix multiplications) followed by activation functions.

As the data flows forward through the network, intermediate outputs, also known as activations or feature maps, are computed at each layer. These activations are then passed on as inputs to the subsequent layers until the final output is produced. The forward propagation process essentially calculates the predicted output of the network for a given input.

Backward Propagation

Backward propagation, short for backward propagation of errors, is the process of computing the gradients of the network's parameters with respect to a loss function. These gradients indicate the sensitivity of the network's output to changes in its parameters and are used to update the parameters during the training process.

The backward propagation algorithm starts from the final output of the network and propagates the error gradients backward through the network's layers. It computes the gradients layer by layer using the chain rule of calculus. The gradients quantify how each parameter contributed to the overall error of the network and provide information on how to adjust the parameters to reduce the error.

Once the gradients are computed, an optimization algorithm (such as stochastic gradient descent) uses them to update the network's parameters in a way that minimizes the loss function. The process of repeatedly performing forward propagation, computing gradients through back-propagation, and updating the parameters is carried out iteratively until the network converges to a desirable level of performance.

4.1.10 Under-fitting vs Over-fitting

Under-fitting is a prevalent challenge in machine learning, occurring when the model fails to establish a meaningful relationship between the input and target variable. Insufficiently capturing the features of the data results in increased errors in both the training and unseen data samples.

Over-fitting is also a prevalent challenge in machine learning but it is really common in deep learning algorithms. Deep learning models try to fit the training data entirely and ends up memorizing the data patterns and the noise/random fluctuations. These models fail to generalize and perform well in the case of unseen data scenarios, defeating the model's purpose.

Figure 4.13 illustrates the under-fitting and over-fitting issue against a good fitting in a classifier problem.

Figure 4.13: *Under-fitting vs Over-fitting vs A Good Fitting. Illustration by towards-datascience*

4.1.11 Strategies to Combat Over-fitting

There are lot of strategies to combat over-fitting, in this section there is presented only those used in the project.

Early Stopping

Early Stopping involves monitoring the model's performance on a validation dataset during training. If the performance stops improving or starts deteriorating, the training process is stopped early. By doing this, early stopping helps find the optimal balance between model complexity and generalization, ensuring the model doesn't become overly specialized to the training data and do not performs well on unseen data.

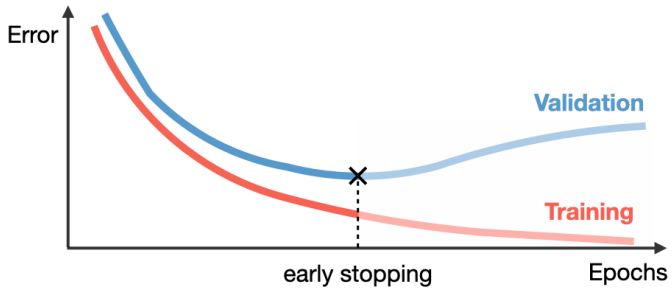


Figure 4.14: *Early Stopping. the vertical doted line represents the point where the early stop mechanism should have been triggered. Illustration by wandb*

Learning Rate Scheduling

The learning rate is a hyper-parameter in machine learning that determines the step size at which a model's parameters are updated during the optimization process. In most optimization algorithms, such as gradient descent, the learning rate controls how quickly or slowly the model learns from the training data.

There are various approaches in learning rate scheduling, bellow you'll find the scheduling used in the thesis.

- **Learning Rate Decay**

  This learning rate scheduling works gradually reducing the learning rate over time or in response to certain conditions. A smaller learning rate allows the model to fine-tune its parameters more cautiously, preventing it from aggressively fitting to noisy or irrelevant patterns in the training data. This smoother adjustment helps the model converge to a better generalize solution, reducing the risk of over-fitting.

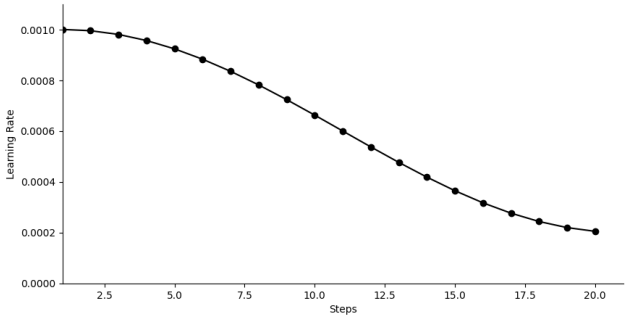


Figure 4.15: *Learning Rate Decay. In this graphic it's represented how the learning rate hyperparameter decrease over the epochs. Illustration by Author*

- **Cycling Learning**

  Cyclic learning rate scheduling is popular for faster convergence and improved generalization. By cycling the learning rate, models explore diverse loss landscape regions, escaping local minima for better optima. Variations like triangular and cosine annealing alternate or smoothly transition between lower and upper bounds. This approach adds variation and exploration to optimization, enhancing model performance and convergence.

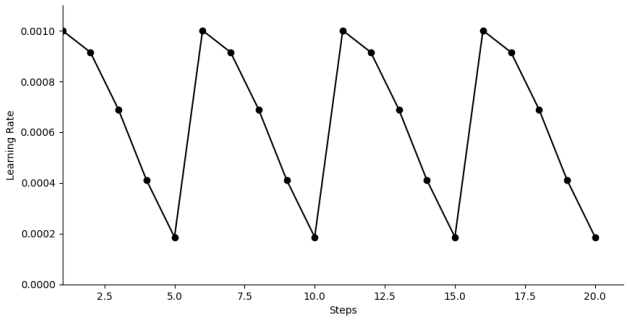
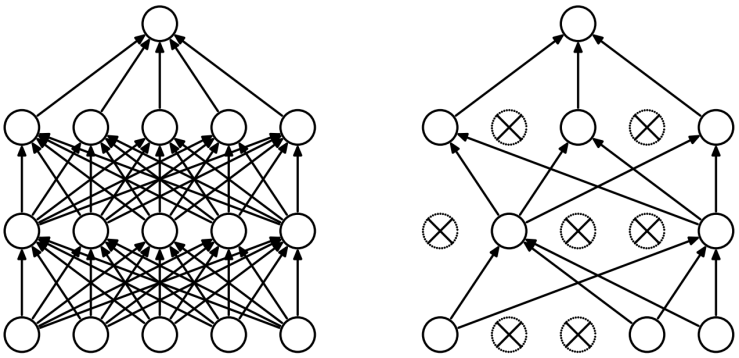


Figure 4.16: *Cosinus Cycling Learning. The graphic show how the learning rate varies following the cosinus functions over the epochs. Illustration by Author*

Dropout

Dropout is a regularization method [Srivastava 2014]. The idea behind dropout is to prevent overfitting and improve generalization in neural networks by randomly "dropping out" or deactivating a fraction of the neurons during each training iteration.

By randomly dropping out neurons, the network becomes more robust and less sensitive to the precise configuration of any single neuron. It forces the network to learn redundant representations and prevents co-adaptation.



(a) Standard Neural Net (b) After applying dropout.

Figure 4.17: *Dropout Neural Net Model.* ***Left****: A standard neural net with 2 hidden layers.* ***Right****: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped. Illustration by Srivastava*

4.1.12 Train, Validation and Test Sets

The primary method employed to validate the models involves dividing the original dataset into three subsets using the Holdout set scheme, see *Figure 4.18*. The percentages applied to divide to the ISIC dataset used for training is 80%, for validation 10% and for testing 10%.

Bellow there are the explanation of each dataset and the propose of all of them.

Training Set

This subset is utilized during the training phase of the model.

Validation Set

This subset was employed to apply trained models to new, unseen examples and evaluate the model's performance. It also aids in selecting the model that best matches the validation set by minimizing the classifier One vs Rest metric in predictions.

Test Set

Comprising non-observed data examples, this set is used to assess the performance of the chosen model. It helps determine if further optimization is required and which techniques should be applied next. Alternatively, it aids in deciding whether to select an alternative model.

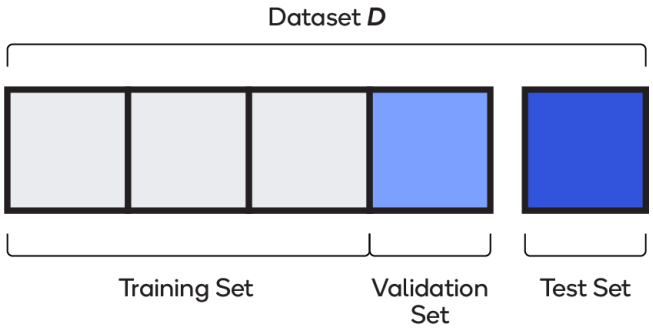


Figure 4.18: *Holdout Test Scheme. Illustration by Qualcomm*

4.1.13 Data Augmentation

Data augmentation encompasses various techniques utilized to expand the dataset by introducing modifications, thereby increasing the number of examples. Its purpose is not only to enlarge the dataset but also to enhance its diversity. By acting as a regularizer, data augmentation aids in mitigating overfitting during machine learning model training.

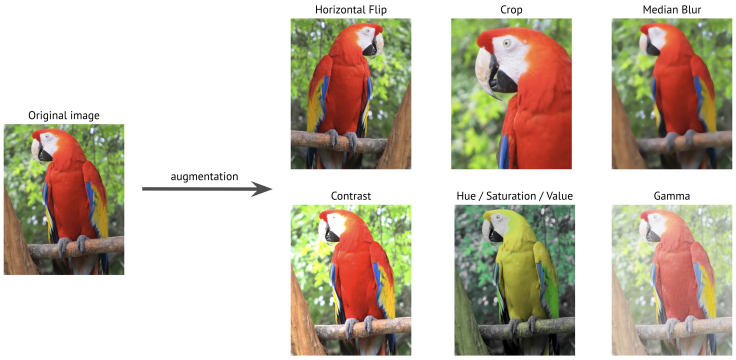


Figure 4.19: *Data Augmentation. The example show the result of applying some transformation to an image. Illustration by Albumentations*

4.1.14 Test-Time Augmentation

Test-time augmentation (TTA) is used during the testing or inference phase of a machine learning model. TTA generates multiple augmented versions of test samples to obtain diverse predictions. By obtaining predictions from these augmented samples and combining them, TTA mimics the behavior of an ensemble of models.

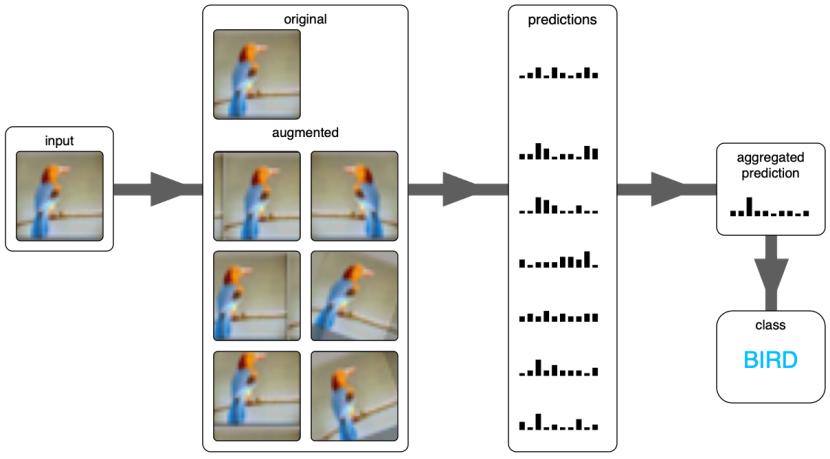


Figure 4.20: *Test-Time Augmentation. Illustration by stepup.ai*

4.1.15 ResNet

ResNet, short for "Residual Network," is a deep convolutional neural network (CNN) architecture that was introduced in 2015 by researchers from Microsoft Research [Kaiming He 2013]. It revolutionized the field of deep learning by addressing the challenge of training very deep neural networks.

The main problem encountered when training deep neural networks is the degradation problem, where the accuracy of the model saturates and then starts to decline rapidly as the network depth increases. This degradation occurs due to the difficulty of optimizing the network's parameters and the vanishing gradient problem, where gradients become extremely small during backpropagation, making it difficult for the network to learn effectively.

ResNet tackled this problem by introducing the concept of residual learning. The key idea is to use "skip connections" or "identity mappings" that allow the network to learn residual functions. Instead of trying to learn the underlying mapping directly, ResNet models learn the residual between the desired mapping and the input.

By introducing these skip connections, ResNet models can effectively "skip" one or more layers during training, allowing information to flow more easily through the network. This mitigates the degradation problem and enables the training of extremely deep networks (e.g., hundreds of layers) without suffering from diminishing accuracy.

In ResNet, each layer of the network contains a residual block. A residual block consists of multiple convolutional layers followed by batch normalization and activation functions. The input to a residual block is added to its output through a skip connection, which directly connects the input to the output. This allows the network to learn the residual information, or the difference between the input and the desired output, making it easier for the network to learn the underlying mapping.

There are various architectural variants or "flavors" of ResNet, including `ResNet-152`, `ResNet-101`, `ResNet-50`, `ResNet-34`, and `ResNet-18`. The number following the name of each ResNet variant indicates the number of inner layers present in the architecture.

As evident from *Figure 4.21*, it can be observed that ResNet architectures with a greater number of layers tend to exhibit higher accuracy. However, it is important to note that this improvement in accuracy comes at the expense of having a significantly larger number of parameters to train.

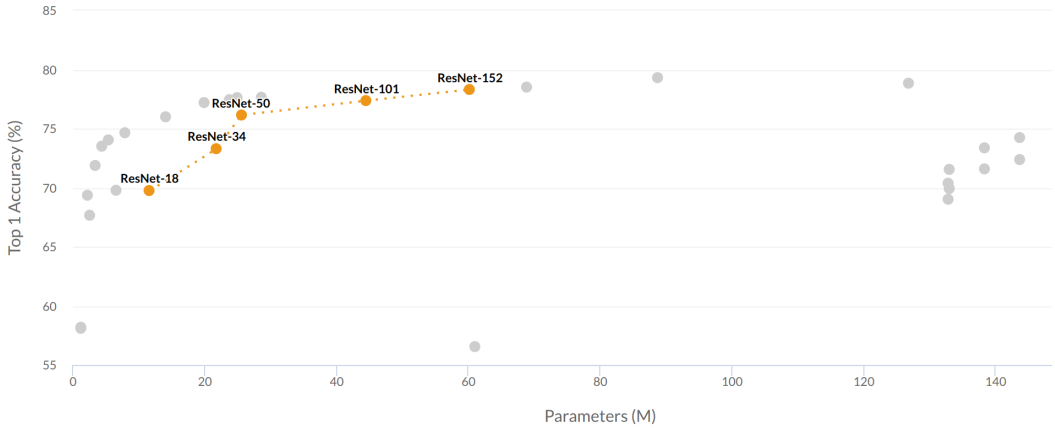


Figure 4.21: *ResNet "flavors" Results on ImageNet. Illustration by paperswithcode*

To accurately assess the performance of each ResNet architecture, refer to the *Table 4.1*, which provides the accuracy achieved on the ImageNet dataset along with the corresponding number of trainable parameters in millions.

Model	**Accuracy**	**Parameters**
ResNet-152	78.31%	60.2M
ResNet-101	77.37%	44.5M
ResNet-50	76.15%	25.6M
ResNet-34	73.30%	21.8M
ResNet-18	69.76%	11.7M

Table 4.1: *Accuracy Achieved on ImageNet and Trainable Parameters of Each ResNet. Each image in the ImageNet dataset is associated with 1 of 1,000 classes.*

4.2 Technical Background

This section provides an introduction to the technical concepts that surround the creation of the CAD system, such as microservices architecture, model exposing, code packaging, containerization, platform deployment.

4.2.1 Microservices Architecture

A microservices architecture is a software design approach that structures an application as a collection of small, independent services, each running in its own process and communicating with each other through lightweight mechanisms. This architecture promotes scalability, modularity, and flexibility.

The present thesis consists of two services: an API and a UI. These services are built using different frameworks and programming languages.

4.2.2 Inference API

The inference API service focuses on handling the back-end logic and data processing. It exposes a well-defined API that allows user to interact with the trained models.

The main end-points that the present thesis API service exposes are explained bellow. The completed source code is not given here, but it can be found in the GitHub repository or in the source code attached to the the thesis.

Consulting Available Models

This end-point returns the list of the available models configured in the configuration file of the API.

```
@app.get("/public_models")
async def public_models():
    """
    Description
    ----------
    Returns the name of the available models
    """
    ...
```

Predict a Single Image

This end-point captures an image and a model identifier creating a unique task and makes the prediction just in time.

```
@app.post("/predict")
async def predict(file: UploadFile = File(...),
                  model_id='vicorobot.8c_b3_768_512_18ep_best_fold0'):
    """
    Description
    ----------
    The function receives a file (expected img with jpeg format)
    then it creates the task that is being returned
    and async does the prediction.
    """
    ...
```

Predict a Jar of Images

This endpoint behaves similarly to the previous endpoint, with the key difference being that it supports predicting a set of images instead of just one. Additionally, the predictions of the images are made in a separate thread, preventing the system from getting stuck due to the time it would take to predict a large number of images.

```
@app.post("/predict_bulk")
async def predict_bulk(bg_tasks: BackgroundTasks,
                       files,
                       model_id='vicorobot.8c_b3_768_512_18ep_best_fold0'):
    """
    Description
    ----------
    Recives a jar of images and then it creates a task
    for this predict that is returned to consult the result
    of the predictions of each img.

    Returns
    -------
    task_id: str
        - folder where the images where saved
    num_files: int
        - number of images saved
```

```
    """
    ...
```

Consult the Predictions

When an inference is made, users need a way to access the results of the requested prediction. To handle this, the inference process stores the predictions and metadata of the models in CSV files within the task directory. These CSV files are then retrieved and combined to generate a response for the client.

```
@app.get("/from_task/{task_id}")
async def from_task(task_id: str):
    """
    Description
    ----------
    Consults the predictions from a task
    """
    ...
```

Exposing the Models

The concept of handling long-running tasks in the background is often implemented to improve the responsiveness and scalability of the system. This approach allows the client to initiate a task and receive a unique identifier, which can later be used to query or retrieve the results of that task (*Figure 4.22*).

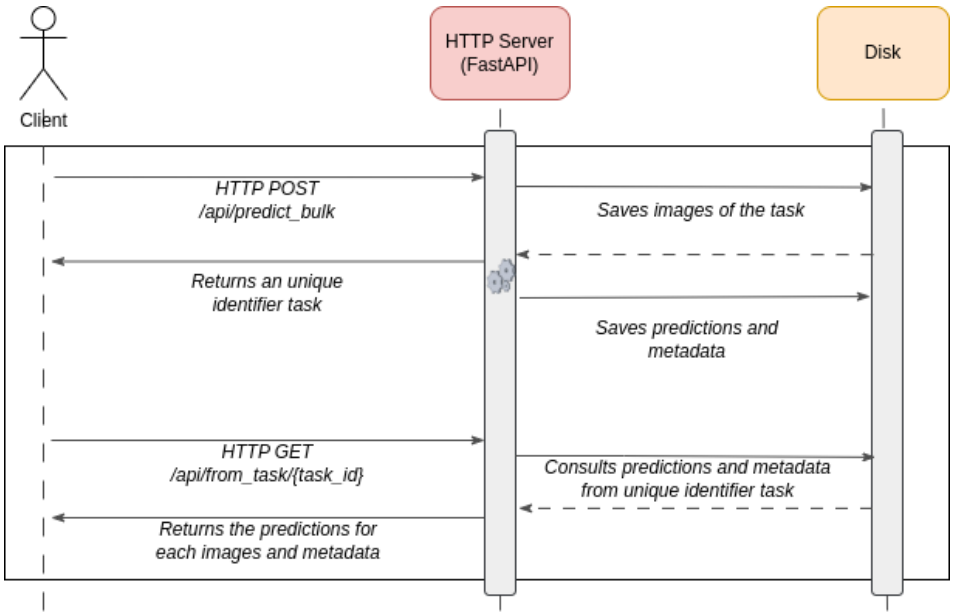


Figure 4.22: *Inferring Images Through the Background Task Mechanism. Notice that in this way we avoid to congest the API thread. Illustration by Author*

When an HTTP task is required, it comes with the identifier of the model that will be used to infer the image or a jar of images. This model is searched for in a volume configuration file located in the API. The configuration file must contain the entry for the model identifier and also include the path to the binary trained model. The trained model is deserialized and used at runtime through the PyTorch API.

Once the path of the model in the operating system is obtained is as easy as load the searilized model throw the Pytorch API as follow:

```
instance_nn = class_nn(out_dim)
checkpoint = torch.load(pytorch_model_path, map_location=device)
model_state_dict = checkpoint['model_state_dict']
instance_nn.load_state_dict(model_state_dict, strict=True)
```

4.2.3 Containerization

Containerization is a technology that allows applications and their dependencies to be packaged into isolated, lightweight containers. These containers provide an encapsulated runtime environment, ensuring that the application runs consistently across different systems.

Containers are created from container images, which are self-contained packages containing the application's code, dependencies, libraries, and configurations. Images serve as the blueprint for creating containers. They can be shared, versioned, and easily deployed on various platforms.

Virtualization frameworks, such as Docker and Podman, provide tools and services to build, manage, and run containers. These frameworks utilize underlying virtualization technologies to create and manage isolated environments. These environments, called containers, offer resource isolation and ensure that applications run consistently across different computing environments.

4.2.4 Platform Deployment

To distribute the containerized API and UI services easily to professionals, there are several solutions available. In this thesis, I propose the creation of a shell script that performs the following steps:

- Creates a directory in the system's home directory to download the source code and artifacts.
- Clones the source code repository from GitHub.
- Moves the encapsulated Python packages into the API source code.
- Builds the API image.
- Builds the UI image.
- Clones the artifacts from GitLab.
- Activates the services using a Docker Compose file, which starts the containers with their configurations based on the previously built images.

By following this approach, professionals can easily distribute the containerized services by running the provided shell script.

CHAPTER 5

Studies and Decisions

This chapter is dedicated to exploring the essential tools required for the project's development and understanding their significance. The initial section of this chapter focuses on delineating the functional and non-functional requirements that the system must fulfill. Subsequently, the second part identifies and elucidates the selected technologies for the project.

5.1 System Requirement

This section explores the necessary requirements for detecting melanoma. It is important to distinguish between the final system and the work involved in its development. Constructing and training a Convolutional Neural Network (CNN) requires substantial computational power, sophisticated libraries, and often a significant amount of time. However, once the model has been trained and saved, it can be readily used. Typically, the response time inferring is within seconds. Yet the inference time of the models is few we also need to take into account the time wasted in the HTTP request and the time wasted in showing the results.

The requirements can be classified into two categories:

- **Functional requirements**

  Functional requirements pertain to the capabilities that the product must possess to fulfill specific user needs.

- **Non-functional requirements**

  Non-functional requirements encompass aspects such as usability, performance, reliability.

Functional requirements

The functional requirements of the system are:

- The objective is to classify whether a single image or a set of images, specifically from a dermoscopy, contains a melanoma or not. The classification task is binary, aiming to determine the presence or absence of melanoma.
- Expose models through an API that can classify whether an image or a set of images contains melanoma or not. The API should not only provide the classification results but also offer additional information such as the specific model used and the probabilities associated with the prediction..
- A user-friendly interface (UI) is implemented to facilitate loading of images and communicate with the API via the HTTP protocol for image inference. The UI should display the results of the selected model for the bulk of loaded images.
- Create images of each service, to distribute the services using container technology.

Non-functional requirements

For training models, Jupyter Notebook, Python3, and Anaconda are used, ensuring compatibility with any computer to run the code and classify images. Sufficient RAM is required to handle the desired images effectively.

Since the API and UI services are containerized, any container management tool such as Docker or Podman can be used, providing the necessary environment to work with containers.

5.2 Hardware

The hardware used for the thesis encompassed a range of machines. The primary development machine employed was a laptop with limited performance. Additionally, a machine provided by Google was utilized within the Google Colab environment for training models that did not require high computational capabilities. Furthermore, for a week of work at the VICOROB laboratory, a separate machine was utilized specifically for training models with more extensive data augmentation requirements, necessitating higher computational resources.

Table 5.1, 5.2, and 5.3 provide an overview of the characteristics of the machines employed for the project. Each of these machines had its own specific use, and it is recommended to select the an appropriate machine based on specific needs.

Development Machine	
OS	Fedora Linux 38
CPU	Intel i5-8250U
GPU	Intel UHD Graphics 620
RAM	8GB
Disk	225GB

Table 5.1: *Development Machine Metrics. (Note: This machine was only used for programming and searching thesis information.)*

Google Machine	
OS	Ubuntu 20.04.6 LTS
CPU	Intel Xeon
GPU	Tesla T4, 16GB
RAM	12GB
Disk	125GB

Table 5.2: *Google Machine Metrics. Please note that this machine was used to train models that take a few hours to accomplish the training.*

VICOROB Machine	
OS	Ubuntu 20.04.2 LTS
CPU	Intel(R) Xeon(R) Silver
GPU	A100, 80GB
RAM	396GB
Disk	6.3T

Table 5.3: *VICOROB Machine Metrics. This machine was used to train models that required high computational resources because of data augmentation.*

5.3 Software

In order to establish and operate the CAD infrastructure, it was crucial to establish separate environments for creating the services and training the models, ensuring that all the required packages were included. The detailed instructions for setting up these distinct environments can be found in *Appendix A*. Additionally, *Appendix B* provides a guide on deploying and activating the services within the CAD infrastructure.

The following pages will outline and provide detailed descriptions of each tool utilized throughout the thesis. These tools played a crucial role in various aspects, including the training process, creation of services such as user interface (UI) and application programming interface (API), deployment of these services with trained models, management of the source code, and comprehensive documentation. Each tool's significance and contribution to the thesis will be thoroughly explained, shedding light on their specific functionalities and importance in the research project.

Python 3.9

Python (*Figure 5.1*) is a high-level, interpreted programming language known for its simplicity and readability. It was created by Guido van Rossum and released in 1991. Its design philosophy emphasizes code readability with the use of significant indentation.

Python is an ideal choice for machine learning and AI-based projects due to several advantages. These include its simplicity and consistency, access to excellent libraries and frameworks specifically designed for AI and ML, flexibility, platform independence, and a large and supportive community.

Figure 5.1: *Python Logo. Illustration by Python Software Foundation*

Anaconda

Anaconda (*Figure 5.2*) is a widely used open-source distribution of the Python programming language. It simplifies the management and deployment of data science and machine learning environments. With Anaconda, you can easily install and manage Python packages using the conda package management system. It also provides tools for creating isolated environments to ensure consistent dependencies and package versions.

Figure 5.2: *Anaconda Logo. Illustration by Anaconda*

Jupyter Notebook

Jupyter Notebook (*Figure 5.3*) is an open-source web application that allows to create and share interactive documents containing live code, visualizations,

and text. It provides an environment for data analysis, visualization, and prototyping. With Jupyter Notebooks, you can write and execute code in individual cells, making it easy to experiment and iterate. It supports multiple programming languages such as Python, R and Haskell.

Figure 5.3: *Jupyter Notebook Logo. Illustration by Jupyter.org*

CUDA

CUDA (*Figure 5.4*), which stands for Compute Unified Device Architecture, is a parallel computing platform and programming model developed by NVIDIA. It enables developers to leverage the power of NVIDIA GPUs (Graphics Processing Units) for high-performance computing tasks.

CUDA has gained widespread adoption in the scientific and computational research communities due to its ability to leverage the computational power of GPUs. It has become an essential tool for accelerating a wide range of applications, from physics simulations and computational biology to data analytics and artificial intelligence.

Figure 5.4: *CUDA Logo. Illustration by Nvidia Corporation*

PyTorch

PyTorch (*Figure 5.5*) is an open-source machine learning framework widely used for deep learning tasks. It provides a flexible and dynamic approach to building

and training neural networks. PyTorch stands out for its integration of GPU acceleration, enabling efficient computations using NVIDIA GPUs.

PyTorch offers a high-level API called torchvision, which simplifies common computer vision tasks such as image classification, object detection, and image generation. It also integrates with other libraries and tools in the Python ecosystem, making it easy to combine PyTorch with popular frameworks like NumPy, SciPy, and pandas.

Figure 5.5: *PyTorch Logo. Illustration by PyTorch.org*

OpenCV

OpenCV (Open Source Computer Vision Library), is a popular open-source computer vision and image processing library. It provides a wide range of functions and algorithms for tasks such as image and video processing, object detection and tracking, feature extraction, and more.

OpenCV (*Figure 5.6*) allows developers to read, write, and manipulate images and videos efficiently. The library offers various image processing functions, including filtering, resizing, color conversion, and geometric transformations.

Figure 5.6: *OpenCV Logo. Illustration by OpenCV Team*

Albumentations

Albumentations (*Figure 5.7*) is an open-source Python library for image augmentation in machine learning and computer vision tasks. It provides a wide range of transformations and optimizations to enhance training data, improving the performance and generalization of machine learning models. With high-performance implementation and seamless integration with popular frameworks,

Albumentations is widely used for efficient and customizable image augmentation.

Figure 5.7: *Albumentations Logo. Illustration by Albumentations Team*

NumPy

NumPy (*Figure 5.8*) is a Python library for efficient numerical computing. It provides a powerful array object for manipulation and computation of multidimensional arrays. With optimized operations and support for broadcasting.

Figure 5.8: *NumPy Logo. Illustration by NumPy Organazation*

Pandas

Pandas (*Figure 5.9*) is a popular Python library for data manipulation and analysis. It provides data structures and functions to efficiently work with structured data, such as tabular data and time series. Pandas' key features include its DataFrame object, which allows for easy handling of data, indexing, filtering, and aggregation operations. With its intuitive and powerful functionality, Pandas is widely used in data preprocessing, exploration, and analysis tasks.

Figure 5.9: *Pandas Logo. Illustration by The pandas development team*

Matplotlib

Matplotlib (*Figure 5.10*) is a widely-used Python library for creating static, animated, and interactive visualizations. It provides a flexible and comprehensive range of functions and tools for generating plots, charts, histograms, and more. Matplotlib allows customization of various aspects of visualizations, including colors, labels, titles, axes, and legends. With its extensive functionality and compatibility with NumPy arrays, Matplotlib is a go-to library for data visualization and presentation in Python.

Figure 5.10: *Matplotlib Logo. Illustration by Matplotlib Organization*

Seaborn

Seaborn (*Figure 5.11*) is a Python data visualization library built on top of Matplotlib. It provides a high-level interface for creating attractive and informative statistical graphics. Seaborn simplifies the process of generating complex visualizations by offering a range of built-in functions for creating appealing plots, such as scatter plots, histograms, bar plots, and heatmaps. Seaborn is widely used for exploratory data analysis and presentation of statistical insights.

Figure 5.11: *Seaborn Logo. Illustration by Seaborn Organization*

WandB

Weights & Biases (*Figure 5.12*) is a machine learning experiment tracking and visualization platform. It offers a suite of tools to track, organize, and analyze machine learning experiments. With wandb, developers can log metrics, hyperparameters, and model checkpoints during training, making it easy to compare and analyze different experiments.

Figure 5.12: *Seaborn Logo. Illustration by Weights & Biases Inc*

FastAPI

FastAPI (*Figure 5.13*) is a modern, high-performance web framework for building APIs with Python. It emphasizes simplicity, speed, and type annotations to create efficient and scalable web applications. FastAPI leverages Python's type hints for automatic request and response validation, enabling faster development and reducing the chances of introducing bugs. It provides asynchronous capabilities using Python's asyncio framework, allowing for high-concurrency and efficient handling of multiple requests. With its intuitive API declaration syntax and automatic generation of interactive documentation, FastAPI simplifies the process of building robust and well-documented APIs.

Figure 5.13: *FastAPI Logo. Illustration by tiangolo*

JS

JavaScript (*Figure 5.14*) is a versatile programming language primarily used for client-side web development. It enables dynamic and interactive functionality on websites, allowing for real-time manipulation of HTML elements, handling user events, and modifying web page content.

Figure 5.14: *JS Logo. Illustration by JS Organization*

SvelteKit

SvelteKit (*Figure 5.15*) is a high-performance JavaScript framework for building web applications. It uses a compiler-based approach, resulting in optimized applications with minimal overhead. It provides built-in routing, supports Server-Side Rendering (SSR) and Static Site Generation (SSG), and promotes component reusability. SvelteKit offers a developer-friendly environment, integrates well with APIs, and generates small bundles for faster loading times.

Figure 5.15: *SvelteKit Logo. Illustration by WIKIPEDIA*

Docker

Docker (*Figure 5.16*) is an open-source platform for building and deploying applications in isolated containers. It simplifies application management, ensures portability across different environments, and provides isolation for enhanced scalability and security. With Docker, developers can package applications and their dependencies into lightweight, self-contained containers, making deployment and scaling more efficient.

Figure 5.16: *SvelteKit Logo. Illustration by WIKIPEDIA*

Podman

Podman (*Figure 5.17*) is designed to manage containers and container images, offering features similar to Docker but with a focus on security and compatibility with industry standards. It allows you to build, run, and manage containers without the need for a separate daemon.

One key advantage of Podman is its rootless mode, which allows users to run containers as non-root, enhancing security and isolation. This eliminates the

need for privileged access, making Podman a preferred choice for environments where running containers as non-root is required.

Figure 5.17: *Podman Logo. Illustration by Red Hat*

Git

Git (*Figure 5.18*) is a distributed version control system used for tracking changes in software projects. It enables collaboration, allows for branching and merging, handles large projects efficiently, and provides robust backup options.

Figure 5.18: *Git Logo. Illustration by Red Hat*

GitHub & GitLab

Both GitHub (*Figure 5.19*) and GitLab (*Figure 5.20*) serve as centralized platforms for hosting and managing Git repositories.

Figure 5.19: *GitHub Logo. Illustration by GitHub*

Figure 5.20: *GitLab Logo. Illustration by GitLab*

LaTeX

LaTeX (*Figure 5.21*) is a typesetting system used for creating high-quality documents, particularly those with mathematical equations and scientific content. It focuses on the structure and formatting of documents, allowing users to concentrate on content.

Figure 5.21: *LaTeX Logo. Illustration by LaTeX*

CHAPTER 6

Planning and Methodology

6.1 Feasibility Study

Before initiating any project, it is of utmost importance to conduct a comprehensive analysis of relevant factors to gauge the likelihood of accomplishing successful outcomes. While estimating the complexity associated with developing, deploying, and maintaining machine learning (ML) systems is indeed a challenging task, it is essential to recognize that this project encompasses not only the research aspect but also significant software and infrastructure development.

This project requires a careful balancing act, as it involves dedicating efforts to both the research and practical implementation aspects. The development of robust software and infrastructure is crucial to support the successful integration of the deep learning models and ensure their efficient deployment and maintenance. This integration can be intricate and demanding, as it requires a deep understanding of both the ML algorithms and the software engineering principles.

In light of these additional dimensions, the study not only aims to evaluate the feasibility of utilizing deep learning techniques but also strives to address the complexities inherent in developing a well-rounded solution. Furthermore, the evaluation will consider the available data and resources to assess the potential of achieving satisfactory results.

6.2 Technical Study

The journey begins with research, a dynamic element that evolves and gains strength through continuous investigation. While it is impossible to guarantee the achievement of all objectives, the way wee see it, there are three fundamental pillars that must be addressed in advance.

Defining the Use Case

A perfect use case outlines a project that is precisely defined, quantifiable, attainable, and possesses clearly identified users and value.

Academic researchers may find satisfaction in uncovering effective solutions that contribute to publications and secure additional funding. Yet, this thesis aims to ensure the successful implementation of a CAD (Computer-Aided Diagnosis) infrastructure specifically designed for classifying melanoma, with the intention of deploying it in real-world scenarios I need to look further than just the pure research.

It is important to note that the scope of this project does not encompass the commercialization of a product or service. Yet the intention of creating services from the primary focus study is present. As a consequence of the use case the technical study is based on analyzing which technology and equipment are necessary for carrying out experiments in the most efficient way.

Equipment and Technical Knowledge

Equipment and technical knowledge are interdependent and collectively contribute to the success of a machine learning project.

In a deep learning project, both the availability of suitable equipment and technical knowledge are essential. The correct equipment, including high performance computing resources and data collection devices, facilitates efficient data handling, prepossessing, model training, and deployment. Simultaneously, technical knowledge plays a vital role in effectively leveraging the available equipment. Understanding diverse machine learning algorithms and methodologies assists in selecting the most appropriate approaches for the problem at hand. Additionally, it is crucial to acknowledge the knowledge required to create other services that accompany the project.

Importance of the Dataset

The success of a project like this one depends not only on the quantity but also on the quality of the data. Data is essential for any deep learning model to learn effectively. Therefore, it is vital to invest time in acquiring an optimal image data-set that possesses sufficient data volume, annotation, truth, and reusabil-

ity.

For computer-based image recognition and analysis, high-quality data is necessary. The objective is to gather data from various patient populations to ensure that the data-set is diverse and accurately represents different disease states and outcomes. Often, healthcare organizations engage medical experts to review and label the data to prevent inaccurate labels and ensure the data-set's significance, as is the case in this project.

6.3 Economical Study

Equipment Cost

Equipment cost pertains to the expenditure incurred on hardware and software licenses necessary for the project.

The hardware and software used in the project are listed in the *Table 6.1*.

Component	Units	Unit Price	Cost
Computer	1	$800	$800
Software (Pytorch, Svelte, FastAPI, etc)	1	$0	$0
Tesla T4, 16GB (GPU)*	1	$2000	$2000
Nvidia A100, 80GB (GPU)*	1	$15000	$15000

Table 6.1: *The Cost of Components. Components marked with (*) were not paid for out of my own pocket; instead, they were borrowed from VICOROB or Google Services.*

Human Resources

In this section, a hypothetical scenario is presented, illustrating multiple work profiles contributing to the project and the associated costs for each service. However, it is important to note that in reality, all tasks were performed by a single individual.

Below, the roles required to achieve the thesis are presented along with their explanations.

- **Data Scientist**

  Data scientists focus on managing and analyzing data throughout the project. They acquire relevant datasets, preprocess the data, develop deep learning models, train and evaluate models, and deploy them in production environments (some cases).

  The average hourly wage for a Data Scientist in United States is **$36** [Salary.com 2021a].

- **Research Scientist**

  Research scientists focus on the exploration and innovation aspects of deep learning projects. Their roles typically include, literature review: Research scientists survey existing literature, stay updated with the latest advancements in deep learning, and identify relevant research papers or techniques that can be applied to the project. Innovation and Experimentation: They contribute to the development of novel deep learning architectures, techniques, or algorithms to address specific project challenges or improve model performance.

  The average hourly wage for a Computer Research Scientist in United States is **$121.163** [Salary.com 2021b].

- **Data Engineer**

  Data engineers are typically in charge of creating the user interface (UI) components of the application that interact with the deep learning models. But the most important tasks handled by this profile would be data pre-processing, API development, and model integration. They would build the necessary APIs to communicate with the deep learning models.

  The average hourly wage for a Data Engineer in United States is **$48** [Salary.com 2021c].

Task	Profile	Time	Cost
Definition and boundaries of the project	Research Scientist	50h	$6,081.5
Data collecting	Data Engineer	10h	$480
Data preprocessing	Data Engineer	10h	$480
Data analysis and visualization	Data Scientist	30h	$1,080
Model development	Research Scientist	60h	$7,297.8
Model training	Data Scientist	80h	$2,880
Model optimization	Data Scientist	120h	$4,320
Models reports	Data Scientist	20h	$720
CAD infrastructure	Data Engineer	220h	$10,560
Infrastructure deployment	Data Engineer	20h	$960
Documentation	Research Scientist	100h	$12,163
Total		**720h**	**$47,022.3**

Table 6.2: *Human Resources Estimated Cost.*

The tasks, along with the assigned profile and estimated cost, are presented in *Table 6.2*.

Methodology

The project methodology employed in this endeavor follows a continuous process that builds upon previous approaches. Additionally, the project incorporates the concept of utilizing idle time effectively. For instance, during the training of models, there are periods of idle time, which I exploit by concurrently working on tasks related to developing the entire infrastructure. This approach allows for maximizing productivity throughout the project. The various stages involved in the methodology are as follows:

1. Identifying the problem and establishing the research objectives.
2. Conducting background research by reviewing prior publications addressing similar or related issues.
3. Choosing a programming language and framework.
4. Study of the characteristics and distribution of the working data-set.
5. Familiarizing oneself with the tools and frameworks will be utilized.The term "equipment cost" pertains to the expenditure incurred on hardware and software licenses necessary for this project.

6. Breaking down the project into smaller tasks.
7. Selecting a specific task.
8. Trying to implement the task in a smart manner.
9. Developing the task.
10. Make GPU do its job for certain number of epochs (Iddle).
    - Choosing or resuming a develop task (API, UI or Virtualization).
    - Developing process.
      - Developing interruption because model is already trained or because early stopped. Go to point 11.
      - If there are remaining tasks, choose a new develop task.
11. Evaluating whether the results meet the expectations.
    - If the results fall short of expectations, return to step 8.
    - If the results meet the expectations, proceed to step 12.
12. Collecting the necessary artifacts and presenting the results in a clear and appealing manner.
13. Deploy models.
14. Deploy the CAD system infrastructure.
15. Write down the report.

Figure 6.1 Illustrates the previous process by making use of a diagram.

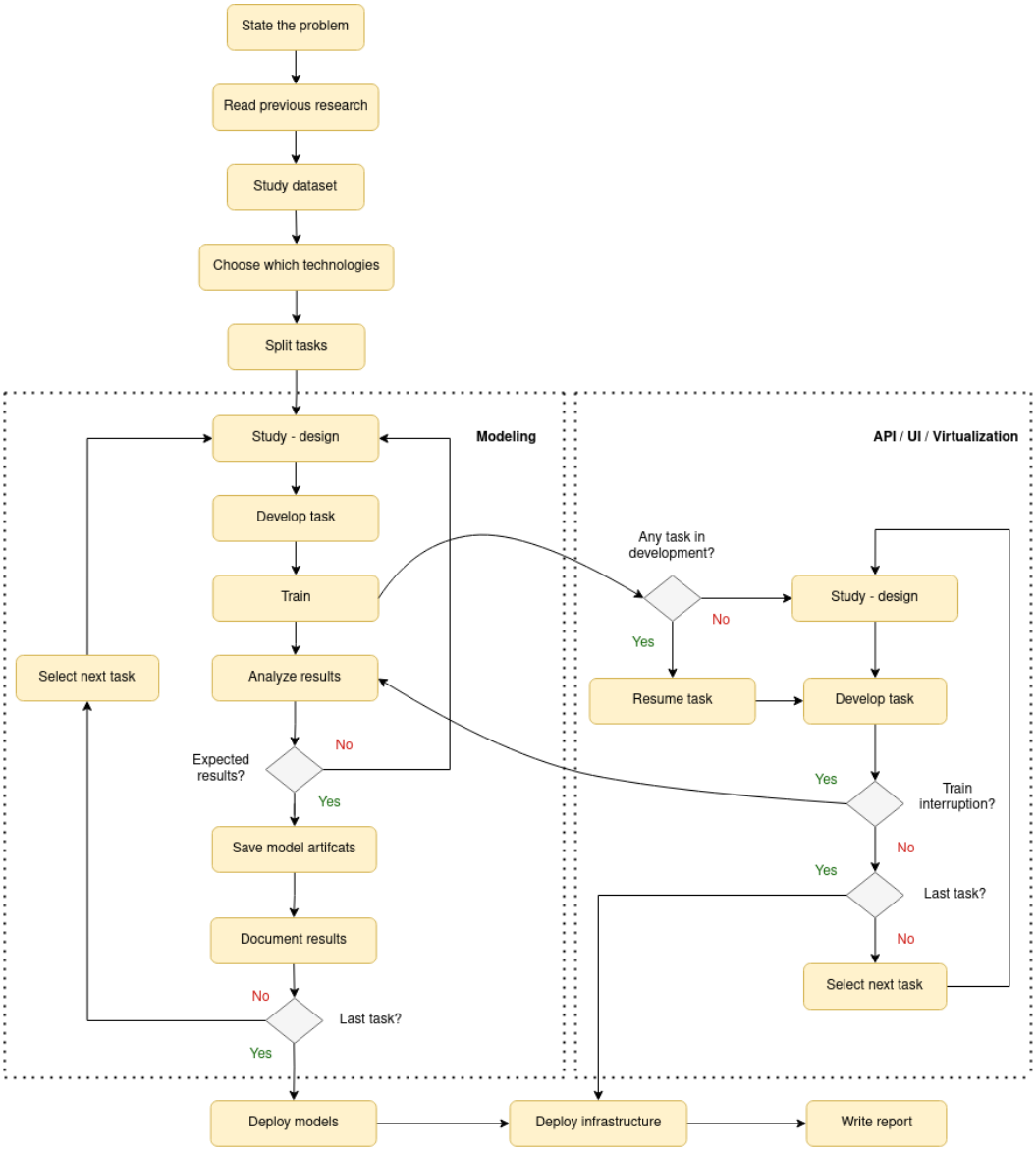


Figure 6.1: *Activity Diagram Describing the Workflow Methodology: Notice that the right workflow is executed every time models are trained and returns to the model evaluation once the models have been trained to analyze results. Illustration by Author*

6.4 Planning

This thesis was developed under an inter-ship in the Accenture SL company, under the category of Research, innovation, and development. The intern-ship was thought to conduct the Master thesis at the University of Girona. We also had the help from VICOROB laboratory which is the Computer Vision and Robotics Research Group of the University of Girona. This thesis runs under the guidance of Dr. Rafel Garcia (UdG) and Luis Llopis (Accenture SL) from 1st of December, 2022 to 12th of July of 2023.

From now on, I present the outline of the activities entailed in this thesis and present an approximate scheduler table to visually depict the allocation of days for their anticipated completion.

6.4.1 Tasks

The task presented here are general, each of this task can be divided in other multiples tasks.

Defining the Boundaries of the Project

Before commencing the actual work, we spent several days engaging in discussions to determine the specific problem we aimed to solve, assess the available data-set, and determine the most suitable technologies to effectively address the problem. This involved researching available melanoma competitions and selecting a technology stack for the project in agreement with the advisors.

Setting Up the Environment

The next weeks was mostly devoted to install and make simple test with the frameworks selected in the previous step. It resulted in setting up the machine with a suitable environment before the development and experimental phase. It should be noted that all other tasks were developed with my personal computer except the training process that was develop using Google Collab service and VICOROB machines.

Study of the Dataset

After setting up the environment, I delved into working with the selected database and carefully examined the distribution of the data-set. It became evident that I was dealing with a multiclass unbalanced data-set, with certain classes being significantly underrepresented compared to others.

Furthermore, I soon realized that due to the substantial size of the data-set, it was not feasible to train the model using any of the devices I had at my disposal. The computational requirements exceeded the capabilities of my available resources.

Searching of Existing Literature

Prior to initiating the actual work, a dedicated few weeks were allocated to conducting an extensive literature review. This crucial step provided valuable insights and a deeper understanding of the proposed research. It involved thoroughly reading academic papers and previous theses that focused on melanoma detection utilizing convolutional neural networks (CNN) or shared a similar research focus.

Acquiring Working Knowledge

Once I had a base knowledge to the problem I face to. I started to follow tutorials per each technology was evolved in the project.

Below is a list of the primary sources I utilized to acquire knowledge about various technologies.

- PyTorch: [mrdbourke 2021] In this course teach you the foundations of machine learning and deep learning with PyTorch. The course is video based.
- SvelteKit: [sveltejs 2021] The official SvelteKit documentation.
- FastAPI: [FastAPI 2021] The official FastAPI documentation.
- Virtualization: To work with virtualization I made use of two container based tools. Docker [Docker 2021] and Podman [Podman 2021] are the tools used to create the images and container of this project.

Modeling and Development

The development process is comprised of various smaller tasks, as depicted in the development flow diagram *Figure 6.1*. During the modeling stage, numerous models were trained using the available data and resources at the time. Additionally, during idle periods of model training, I focused on other aspects of the project. This involved creating a user interface (UI) specifically designed for professionals to interact with, as well as developing an API that is independent of the UI. The API facilitates image loading and inference using the selected model. Furthermore, extensive testing was conducted to deploy these services and configure individual volumes for optimal performance.

Deployment

After developing the CAD infrastructure and uploading the source code to a private GitHub repository, and training the models, I proceeded to upload the configurations and trained models to a public GitLab repository at https://gitlab.com/wilberquito/open.thesis. They can be easily downloaded using Git[1]. However, it's important to note that deployment is restricted to entities recognized by my account. Authentication is done using an SSH Key, which provides secure authentication in SSH connections. It's worth mentioning that the source code will not be publicly available at this time.

The final step of the deployment process involved creating a Shell Script. This script is responsible for downloading the source code from GitHub, as well as retrieving the trained models and configurations from GitLab. It then utilizes the configuration files specific to each service to create the necessary images and instantiate the containers accordingly.

Writing the Report

While certain chapters were written throughout the process, the last month proved to be crucial for completing the final chapters, expand the chapters already written and incorporating any recommended changes suggested by the advisors.

[1] Git is version control software designed by Linus Torvalds

6.4.2 Timeline

Task	Time	Color
Definition and boundaries of the project	40h	
Setting up the environment	20h	
Study of the data-set	20h	
Searching of existing literature	60h	
Acquiring working knowledge	180h	
Modeling and development	240h	
Deployment	40h	
Writing the report	120h	
Total	**720h**	

Table 6.3: *Estimated Hours per Task.*

tasks/weeks	0	3	6	9	12	15	18	21	24	27	30
T1											
T2											
T3											
T4											
T5											
T6											
T7											
T8											

Table 6.4: *Estimated Timeline. All tasks has been scheduled in a timeline that starts from the first week to last the week. The time window is three weeks.*

CHAPTER 7

Methodological Contribution

This chapter is devoted to analyzing which parts of the project are of deep importance, such as the distribution of the data or which hyperparameters must be taken into account for training, and to designing a diagram or solution that provides answers to these matters. In this chapter, we also examine the reasons behind the selection of the base model for transfer learning and the methods used to regularize these trained models.

7.1 Data Analysis

As previously mentioned, the data is acquired from the ISIC Archive, specifically from the SIIM-ISIC Melanoma Classification competition available on Kaggle [ISIC 2020]. This challenge's dataset is a subset of the larger ISIC Archive and includes data from 2019 and 2020.

For the thesis, we opted to use images with a resolution of 512x512 pixels, despite the availability of higher resolutions such as 768x768 and 1024x1024. The decision to select a lower resolution was purely technical, as higher resolution images would require greater computational resources.

For training the models, we utilized 8 classes selected from the original set, as the remaining classes were considered residuals. Any samples that were not categorized as one of the following classes were excluded from the training process.

- melanoma
- nevus
- BCC (Basal Cell Carcinoma)
- BKL (Benign lesions of the keratosis)
- AK (Actinic Keratosis)
- SCC (Squamous Cell Carcinoma)

- VASC (Vascular Lesions)
- DF (Dermatofibroma)

The filtered dataset consists of 31,265 different image samples with an unbalanced distribution across all classes (see Figure 7.1).

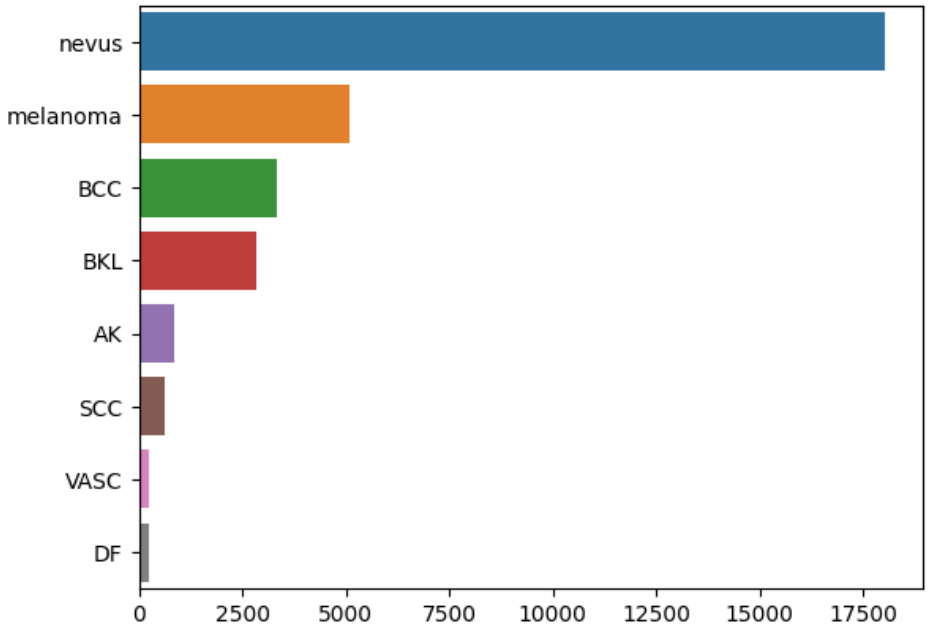


Figure 7.1: *Filtered Dataset Distribution. Having an unbalanced dataset is challenging during training. Models tend to overfit to the more frequent classes since targeting the majority class is easier. Illustration by Author*

7.2 Stratification

Unbalanced datasets present a significant challenge in machine learning problems. Various methods can be employed to address this issue, including oversampling, under-sampling, and more. In our approach, we decided to tackle this problem by utilizing stratification in the train, validate, and test datasets. This ensures that all datasets maintain the same distribution of classes, even though they are unbalanced.

To address the issue of unbalanced distribution, we created a function that made use of the `stratify` parameter in the scikit-learn's `train_test_split` function, which is designed for such scenarios. By default, this parameter is set to the labeled prediction, ensuring that the resulting train, validate and test datasets maintain a proportional representation of the different classes in the data.

```
def train_validate_split(df: pd.DataFrame,
                         random_state: int = 42,
                         validate_size: int = 0.25):
    """Split dataframe into random train and validate dataframe,
    it uses stratify thecnique because of the unbalanced dataset"""

    X_train, X_val = train_test_split(df,
                                      random_state=random_state,
                                      train_size=(1-validate_size),
                                      stratify=df['target'])
    X_train = pd.DataFrame(X_train)
    X_train.columns = df.columns
    X_val = pd.DataFrame(X_val)
    X_val.columns = df.columns

    return X_train, X_val
```

Creating the train, validation and test dataset was as easy as follow,

```
train_df, validate_df = m_dataset.train_validate_split(df,
                                                       random_state=42,
                                                       validate_size=0.2)

validate_df, test_df = m_dataset.train_validate_split(validate_df,
                                                      random_state=42,
                                                      validate_size=0.5)
```

Please note that the training dataset comprises 80% of the filtered dataset, which corresponds to a total of 25,012 samples. In contrast, both the validation and test datasets are composed of 10% of the filtered dataset, amounting to 3,126 samples each.

7.3 Data Augmentation

As it was explained in *Chapter 4*, data augmentation is a technique that creates more data by applying multiple transformations on an image. The main goal of data augmentation is to increase the volume, quality and diversity of training data.

For the thesis, the python package `albumentations` was use for this goal, the package support a big variety of transformations.

Horizontal and vertical flips

An image flip means reversing the rows or columns of pixels in the case of a vertical or horizontal flip respectively.

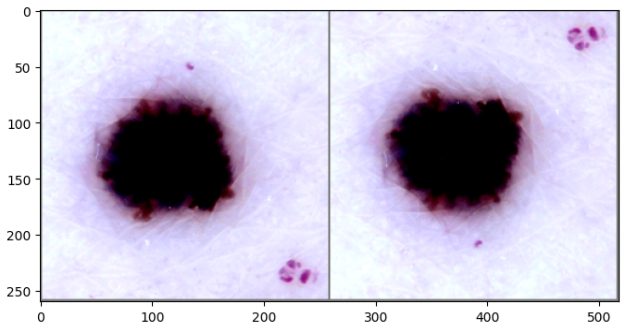


Figure 7.2: *Horizontal flip.* ***Left****: original image,* ***Right****: augmented image. Illustration by Author*

Gaussian blur

Blur the input image using a Gaussian filter with a random kernel size.

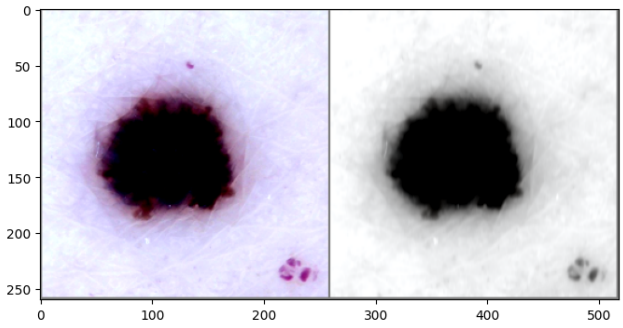


Figure 7.3: *Gaussian Blur.* ***Left****: original image,* ***Right****: augmented image. Illustration by Author*

Random brightness contrast

Randomly change brightness and contrast of the input image.

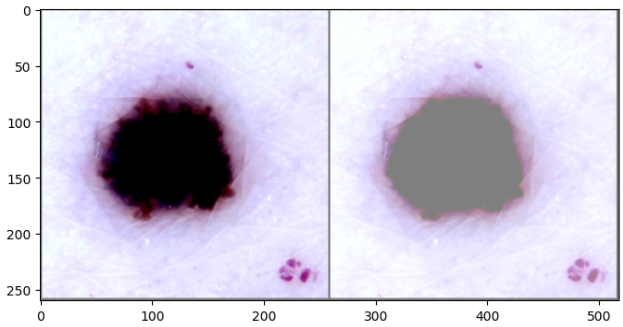


Figure 7.4: *Random Brightness Contrast.* ***Left****: original image,* ***Right****: augmented image. Illustration by Author*

As a result of applying the following data augmentation pipeline,

```
transforms_train = A.Compose([
    A.Transpose(p=0.5),
    A.VerticalFlip(p=0.5),
    A.HorizontalFlip(p=0.5),
    A.RandomBrightnessContrast(contrast_limit=0.2, p=0.75),
    A.OneOf([
        A.MotionBlur(blur_limit=5),
        A.MedianBlur(blur_limit=5),
        A.GaussianBlur(blur_limit=5),
        A.GaussNoise(var_limit=(5.0, 30.0)),
    ], p=0.7),
    A.OneOf([
        A.OpticalDistortion(distort_limit=1.0),
        A.GridDistortion(num_steps=5, distort_limit=1.),
        A.ElasticTransform(alpha=3),
    ], p=0.7),
    A.CLAHE(clip_limit=4.0, p=0.7),
    A.HueSaturationValue(hue_shift_limit=10,
                         sat_shift_limit=20,
                         val_shift_limit=10,
                         p=0.5),
    A.ShiftScaleRotate(shift_limit=0.1,
                       scale_limit=0.1,
                       rotate_limit=15,
                       border_mode=0,
```

```
                        p=0.85),
        A.Resize(image_size, image_size),
        A.CoarseDropout(max_holes=1,
                        max_height=int(image_size * 0.375),
                        max_width=int(image_size * 0.375),
                        p=0.7),
        A.Normalize(mean=mean, std=std)
    ])
```

The train dataset (*Figure 7.5*),

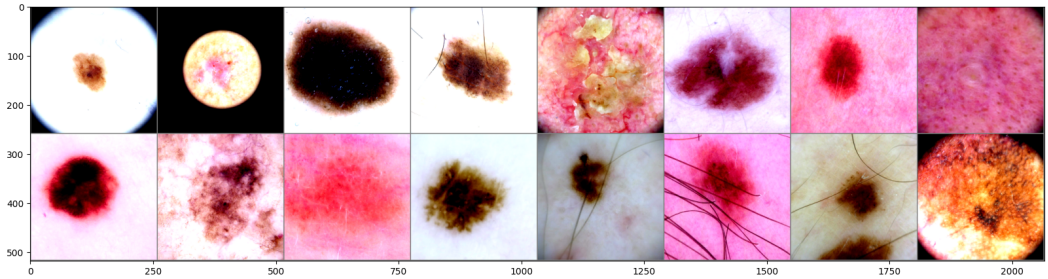


Figure 7.5: *Random sample of images in the train dataset. Illustration by Author*

Is mapped into an augmented train dataset (*Figure 7.6*).

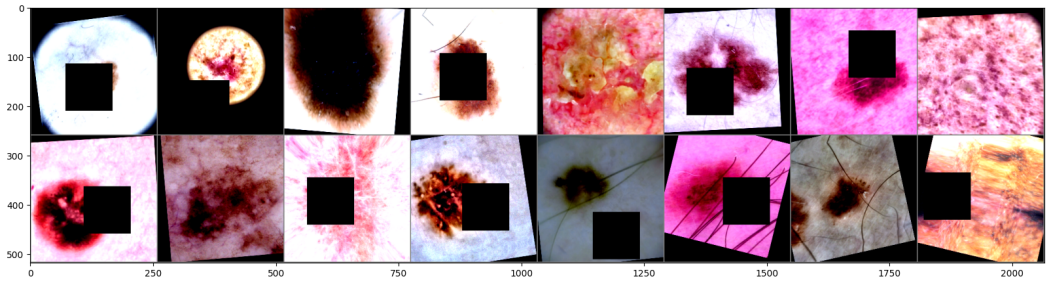


Figure 7.6: *Augmented random sample of images in the train dataset. Illustration by Author*

7.4 ResNet-18 as Base Model

We utilized the pre-trained weights of the `ResNet-18` as the base model for transfer learning. As described in *Chapter 4*, this model achieves an accuracy of approximately 69.76% on the `ImageNet` dataset, which may not be as high as some other models. However, due to its relatively small number of trainable parameters (11.7 million), we decided to select this model over other available options.

In ResNet architecture (Figure 7.7), each layer of the network contains a residual block (BasicBlock).

```
==========================================================================================================
Layer (type:depth-idx)                   Input Shape               Output Shape              Param #
==========================================================================================================
ResNet                                   [1, 3, 256, 256]          [1, 1000]                 --
├─Conv2d: 1-1                            [1, 3, 256, 256]          [1, 64, 128, 128]         9,408
├─BatchNorm2d: 1-2                       [1, 64, 128, 128]         [1, 64, 128, 128]         128
├─ReLU: 1-3                              [1, 64, 128, 128]         [1, 64, 128, 128]         --
├─MaxPool2d: 1-4                         [1, 64, 128, 128]         [1, 64, 64, 64]           --
├─Sequential: 1-5                        [1, 64, 64, 64]           [1, 64, 64, 64]           --
│    └─BasicBlock: 2-1                   [1, 64, 64, 64]           [1, 64, 64, 64]           73,984
│    └─BasicBlock: 2-2                   [1, 64, 64, 64]           [1, 64, 64, 64]           73,984
├─Sequential: 1-6                        [1, 64, 64, 64]           [1, 128, 32, 32]          --
│    └─BasicBlock: 2-3                   [1, 64, 64, 64]           [1, 128, 32, 32]          230,144
│    └─BasicBlock: 2-4                   [1, 128, 32, 32]          [1, 128, 32, 32]          295,424
├─Sequential: 1-7                        [1, 128, 32, 32]          [1, 256, 16, 16]          --
│    └─BasicBlock: 2-5                   [1, 128, 32, 32]          [1, 256, 16, 16]          919,040
│    └─BasicBlock: 2-6                   [1, 256, 16, 16]          [1, 256, 16, 16]          1,180,672
├─Sequential: 1-8                        [1, 256, 16, 16]          [1, 512, 8, 8]            --
│    └─BasicBlock: 2-7                   [1, 256, 16, 16]          [1, 512, 8, 8]            3,673,088
│    └─BasicBlock: 2-8                   [1, 512, 8, 8]            [1, 512, 8, 8]            4,720,640
├─AdaptiveAvgPool2d: 1-9                 [1, 512, 8, 8]            [1, 512, 1, 1]            --
├─Linear: 1-10                           [1, 512]                  [1, 1000]                 513,000
==========================================================================================================
```

Figure 7.7: *Resume of the ResNet-18 Architecture. Notice that the default output dimension of ResNet-18 is 1000. Illustration by Author*

A residual block consists of multiple convolutional layers followed by batch normalization and activation functions (*Figure 7.8*).

```
└─BasicBlock: 2-1
│    └─Conv2d: 3-1
│    └─BatchNorm2d: 3-2
│    └─ReLU: 3-3
│    └─Conv2d: 3-4
│    └─BatchNorm2d: 3-5
│    └─ReLU: 3-6
```

Figure 7.8: *ResNet18 Residual Block. Illustration by Author*

The input to a residual block is added to its output through a skip connection, which directly connects the input to the output (*Figure 7.9*). This allows the

network to learn the residual information, or the difference between the input and the desired output, making it easier for the network to learn the un- derlying mapping.

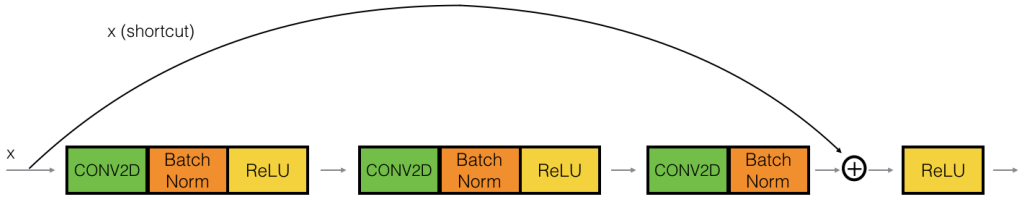


Figure 7.9: *ResNet Skip Connection. Illustration by medium*

There are many ways to use `ResNet-18` as base transfer learning models. We made use of the PyTorch API to accomplish it.

```
from torchvision.models import (resnet18,
                                ResNet18_Weights)
net = resnet18(weights=ResNet18_Weights.IMAGENET1K_V1)
```

7.5 Proposed Models

As explained in the section before, the base model is the `ResNet-18`. This model classify between 1000 classes. But our goal is to classify between 8 different classes.

The very first approach to solve this issue was to encapsulate the ResNet into a `Module` class, which is a class that can be trained in the PyTorch framework.

```
class ResNet18_Melanoma(nn.Module):

    def __init__(self, out_features: int):
        super(ResNet18_Melanoma, self).__init__()

        self.net = resnet18(weights=ResNet18_Weights.IMAGENET1K_V1)
        in_features = self.net.fc.in_features
        self.net.fc = nn.Identity()
        self.myfc = nn.Linear(in_features, out_features)

    def transfer(self, x: torch.Tensor):
        return self.net(x)
```

```
    def forward(self, x: torch.Tensor):
        x = self.transfer(x).squeeze(-1).squeeze(-1)
        x = self.myfc(x)
        return x
```

As you can see above, we utilize the weights of the trained `ResNet-18` and save the trained model into an instance of the `ResNet18_Melanoma` class. Then, we retrieve the instance of the `ResNet-18` and modify its fully connected layer by assigning it to an Identity layer, which doesn't perform any operations. Finally, we instantiate a new fully connected layer with the expected output size. In this case, the expected output size is 8, representing the number of classes. See (*Figure 7.10*).

```
==========================================================================================================
Layer (type:depth-idx)                        Input Shape               Output Shape              Param #
==========================================================================================================
ResNet18_Melanoma                             [1, 3, 256, 256]          [1, 8]                    --
├─ResNet: 1-1                                 [1, 3, 256, 256]          [1, 512]                  --
│    └─Conv2d: 2-1                            [1, 3, 256, 256]          [1, 64, 128, 128]         9,408
│    └─BatchNorm2d: 2-2                       [1, 64, 128, 128]         [1, 64, 128, 128]         128
│    └─ReLU: 2-3                              [1, 64, 128, 128]         [1, 64, 128, 128]         --
│    └─MaxPool2d: 2-4                         [1, 64, 128, 128]         [1, 64, 64, 64]           --
│    └─Sequential: 2-5                        [1, 64, 64, 64]           [1, 64, 64, 64]           147,968
│    └─Sequential: 2-6                        [1, 64, 64, 64]           [1, 128, 32, 32]          525,568
│    └─Sequential: 2-7                        [1, 128, 32, 32]          [1, 256, 16, 16]          2,099,712
│    └─Sequential: 2-8                        [1, 256, 16, 16]          [1, 512, 8, 8]            8,393,728
│    └─AdaptiveAvgPool2d: 2-9                 [1, 512, 8, 8]            [1, 512, 1, 1]            --
│    └─Identity: 2-10                         [1, 512]                  [1, 512]                  --
├─Linear: 1-2                                 [1, 512]                  [1, 8]                    4,104
==========================================================================================================
```

Figure 7.10: *Resume of the ResNet18_Melanoma Architecture. Notice that the output dimension of ResNet18_Melanoma is 8. Illustration by Author*

During the model training process, the `forward` function is triggered with images represented as a tensor. It first performs transfer learning using the base model through the transfer function. Then, it utilizes the output of the transfer process to train the new fully connected layer.

In the evaluation and analyse stage of the different training approaches of the `ResNet18_Melanoma` we observed significant over-fitting. To address this issue, we propose the `ResNet18_Dropout_Melanoma` model, which follows the same behavior as the previous model but includes a regularization layer that utilizes the average of 5 dropout layers after the transfer learning process. This regularization layer helps to mitigate the over-fitting problem found in the previous model.

```
class ResNet18_Dropout_Melanoma(nn.Module):
    """This model is based on ResNet18 but uses the
    average of the result of a list of dropout layers
    to avoid overfitting"""
```

```
def __init__(self, out_features: int):
    super(ResNet18_Dropout_Melanoma, self).__init__()

    self.net = resnet18(weights=ResNet18_Weights.IMAGENET1K_V1)
    in_features = self.net.fc.in_features
    self.net.fc = nn.Identity()
    self.myfc = nn.Linear(in_features, out_features)
    self.dropouts = nn.ModuleList([nn.Dropout(0.5) for _ in range(5)])

def transfer(self, x: torch.Tensor):
    return self.net(x)

def forward(self, x: torch.Tensor):
    x = self.transfer(x).squeeze(-1).squeeze(-1)

    for i, dropout in enumerate(self.dropouts):
        if i == 0:
            out = self.myfc(dropout(x))
        else:
            out += self.myfc(dropout(x))

    out /= len(self.dropouts)
    return out
```

We provide the resume architecture of `ResNet18_Dropout_Melanoma` model in *Figure 7.11*. The `recursive` label in the provided image means that the layers have been "forwarded" multiple times instead of constructed multiple times.

```
==========================================================================================================================
Layer (type:depth-idx)                        Input Shape               Output Shape              Param #
==========================================================================================================================
ResNet18_Dropout_Melanoma                     [1, 3, 256, 256]          [1, 8]                    --
├─ResNet: 1-1                                 [1, 3, 256, 256]          [1, 512]                  --
│    └─Conv2d: 2-1                            [1, 3, 256, 256]          [1, 64, 128, 128]         9,408
│    └─BatchNorm2d: 2-2                       [1, 64, 128, 128]         [1, 64, 128, 128]         128
│    └─ReLU: 2-3                              [1, 64, 128, 128]         [1, 64, 128, 128]         --
│    └─MaxPool2d: 2-4                         [1, 64, 128, 128]         [1, 64, 64, 64]           --
│    └─Sequential: 2-5                        [1, 64, 64, 64]           [1, 64, 64, 64]           147,968
│    └─Sequential: 2-6                        [1, 64, 64, 64]           [1, 128, 32, 32]          525,568
│    └─Sequential: 2-7                        [1, 128, 32, 32]          [1, 256, 16, 16]          2,099,712
│    └─Sequential: 2-8                        [1, 256, 16, 16]          [1, 512, 8, 8]            8,393,728
│    └─AdaptiveAvgPool2d: 2-9                 [1, 512, 8, 8]            [1, 512, 1, 1]            --
│    └─Identity: 2-10                         [1, 512]                  [1, 512]                  --
├─ModuleList: 1-10                            --                        --                        --
│    └─Dropout: 2-11                          [1, 512]                  [1, 512]                  --
├─Linear: 1-3                                 [1, 512]                  [1, 8]                    4,104
├─ModuleList: 1-10                            --                        --                        --
│    └─Dropout: 2-12                          [1, 512]                  [1, 512]                  --
├─Linear: 1-5                                 [1, 512]                  [1, 8]                    (recursive)
├─ModuleList: 1-10                            --                        --                        --
│    └─Dropout: 2-13                          [1, 512]                  [1, 512]                  --
├─Linear: 1-7                                 [1, 512]                  [1, 8]                    (recursive)
├─ModuleList: 1-10                            --                        --                        --
│    └─Dropout: 2-14                          [1, 512]                  [1, 512]                  --
├─Linear: 1-9                                 [1, 512]                  [1, 8]                    (recursive)
├─ModuleList: 1-10                            --                        --                        --
│    └─Dropout: 2-15                          [1, 512]                  [1, 512]                  --
├─Linear: 1-11                                [1, 512]                  [1, 8]                    (recursive)
==========================================================================================================================
```

Figure 7.11: *Resume of the ResNet18_Dropout_Melanoma Architecture. Notice that the output dimension of ResNet18_Dropout_Melanoma is 8. Illustration by Author*

You can instantiate any of these models with the necessary `out_features` by following the code snippet below:

```
out_features = len(mapping)
model = m_models.ResNet18_Dropout_Melanoma(out_features)
```

Please note that the variable `out_features` should be defined before doing any instance.

7.6 Train-Validate Loop

```
def train_model(model: nn.Module,
                mel_idx: int,
                about_data: Dict,
                device: torch.device,
                criterion: nn.Module,
                optimizer: torch.optim.Optimizer,
                scheduler: torch.optim.lr_scheduler.LRScheduler = None,
                num_epochs: int = 25,
                patience: int = 5,
                writter: Writter = None,
                val_times: int = 1):
. . .
```

Among other features, the train validate loop supports:

- Train the model using the training dataset.
- Validate the model using the validation dataset.
- Modify the optimizer's behavior using a scheduler.
- Implement an early stop mechanism by defining the `patience` argument.
- Save the weights of the trained model and evaluate metrics using a higher-order function.
- Apply test-time augmentation during the validation phase, repeating it a certain number of times using the `val_times` argument.

7.7 Writer Callback

PyTorch does not provide built-in wrappers to automatically save model performance during the training and validation phases. However, it is essential to have information on how well a model performs after each epoch in order to compare different training approaches.

To address this issue, we have implemented a function that can save the model weights after each epoch and maintain a history of metrics from both the training and validation phases. Additionally, we log the training process by sending the latest state of the metrics using the Weight and Biases MLOPS service. This allows us to track and analyze the training progress effectively

```
def model_writter(model_name: str):
    """Saves the pythorch trainned model and generates
    a log file in csv format with the current trainning
    and validation phases. It finally send the last log
    to wand service."""

    def writter(point: Dict):
        # Save checkpoint
        m_checkpoint.save_checkpoint(point,
                                     model_name + '.pth.tar')

        # Logging the trainning
        log_filename = model_name + '.csv'
        stats = point['stats']
```

```
        pd.DataFrame(stats).to_csv(log_filename)

        # wand logging
        wandb.log({
            "train_acc": stats["train_acc"][-1],
            "train_loss": stats["train_loss"][-1],
            "train_ovr": stats["train_ovr"][-1],
            "val_acc": stats["val_acc"][-1],
            "val_loss": stats["val_loss"][-1],
            "val_ovr": stats["val_ovr"][-1],
        })

    return writter
```

7.8 Test-Time Augmentation

Test-time augmentation (TTA), as explained in Chapter 4, is an ensemble mechanism. TTA is not only useful for inferring the test set but also for inferring the validation set. Since the validation set does not undergo any data augmentation during the prediction phase, we apply multiple dummy transformations and average the predictions to obtain the final prediction. This technique helps improve the robustness and reliability of the model's predictions on unseen data.

```
@torch.inference_mode()
def test_time_augmentation(model: torch.nn,
                           inputs: torch.Tensor,
                           val_times: int):
    """Applies time test time agumentation to a
    set of tensor images an returns the logits"""

    model.eval()

    logits = []
    for n in range(val_times):
        augmented_img = test_time_transform(inputs, n)
        augmented_img = torch.unsqueeze(augmented_img, 0)
        outputs = model(test_time_transform(inputs, n))
        logits.append(outputs)

    stacked_logits = torch.stack(logits)
    stacked_logits = torch.mean(stacked_logits, dim=0)
```

```
    return stacked_logits


def test_time_transform(img: torch.Tensor, n: int):
    """Given a tensor it applies dummy
    transformation on de n value"""

    if n >= 4:
        return img.transpose(2, 3)
    if n % 4 == 0:
        return img
    elif n % 4 == 1:
        return img.flip(2)
    elif n % 4 == 2:
        return img.flip(3)
    elif n % 4 == 3:
        return img.flip(2).flip(3)
```

7.9 Training Models

We could train eight models with different training approaches in various machine environments. All of these models are based on transfer learning using the `ResNet-18` architecture and fine-tuning this architecture.

The training information, including the training environment and hyperparameters, for each model is provided in Table 7.2. To facilitate understanding, we have also included a name mapping in Table 7.1 for easy reference.

Name	Mapping
ResNet18_V0	M0
ResNet18_V1	M1
ResNet18_V2	M2
ResNet18_V3	M3
ResNet18R_V0	M4
ResNet18R_V1	M5
ResNet18R_V2	M6
ResNet18R_V3	M7
ResNet18_Melanoma	R18M
ResNet18_Dropout_Melanoma	R18DM
CosineAnnealingWarmRestarts	CAWR
CosineAnnealingLR	CALR
StepLR	SLR
Tesla T4, 16GB (GPU)	TT4
Nvidia A100, 80GB (GPU)	NA100

Table 7.1: *Mapping of Names. This table is provided for an easy visualization for next tables.*

	M0	M1	M2	M3	M4	M5	M6	M7
Model	R18M	R18M	R18M	R18M	R18DM	R18DM	R18DM	R18DM
Epochs	20	20	20	20	40	40	40	40
Batch Size	400	400	400	400	1024	1024	1024	1024
Optimizer	SGD	SGD	SGD	SGD	SGD	SGD	SGD	SGD
Learning Rate	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001
Scheduler		SLR	CALR	CAWR		SLR	CALR	CAWR
Data Augmentation	No	No	No	No	Yes	Yes	Yes	Yes
Dropout Regularization	No	No	No	No	Yes	Yes	Yes	Yes
GPU	TT4	TT4	TT4	TT4	NA100	NA100	NA100	NA100
Training Time	1h 45m	1h 22m	1h 43m	1h 38m	1d 7h 30m	1d 7h 4m	1d 7h 1m	1d 12h 55m

Table 7.2: *Training Information For Each Model. Empty spaces represent non-use of that feature.*

7.10 Pipeline

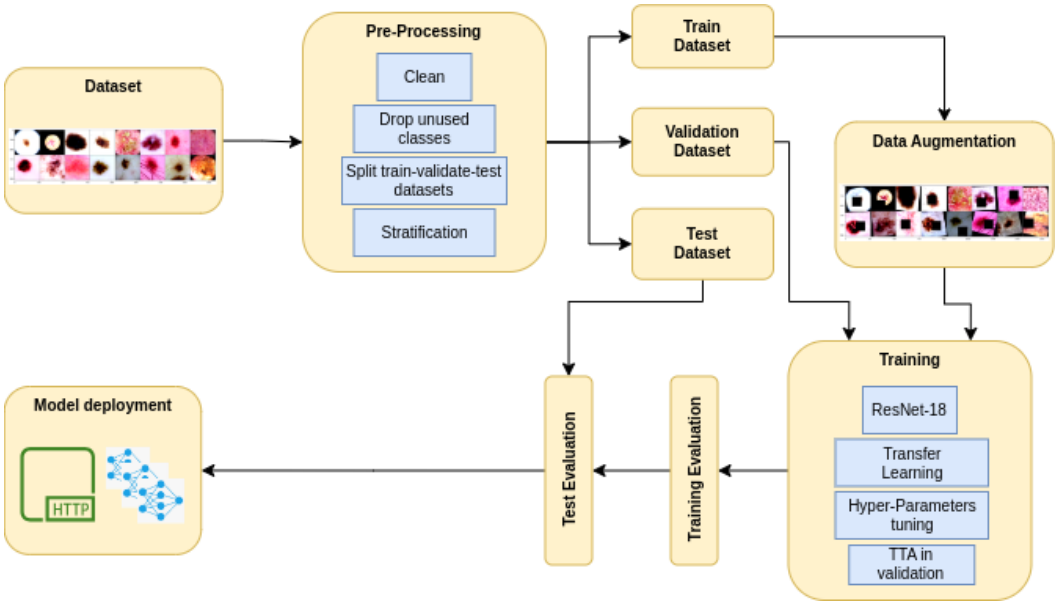


Figure 7.12: *Training And Exposing CAD Infrastructure models. Illustration by Author*

Figure 7.12 illustrates the pipeline used for training and deploying models. The pipeline consists of four stages.

The first stage involves cleaning and splitting the initial dataset into smaller datasets. This step ensures that the data is organized and ready for further processing.

The second stage focuses on training and validating the models using the training and validation datasets. During this stage, the system reads images and applies data augmentation techniques to train images and Test Time Augmentation (TTA) to validation images. These techniques enhance the model's performance by introducing variations in the data and improving its generalization ability.

The third stage involves analyzing the training results obtained from different training approaches. In this section, we evaluate and analyze the model's performance by comparing the predicted results against the test dataset. This step helps us understand how well the models are learning and performing on unseen data.

The last stage revolves around exposing the trained models through an API's container image. This container image allows for easy deployment and integration of the models into other systems or applications, providing a convenient way to utilize the trained models for various tasks.

Overall, the pipeline depicted in *Figure 7.12* showcases a systematic approach to training models, evaluating their performance, and making them available for consumption through an API infrastructure.

Chapter 8

Results

CHAPTER 9

Conclusions and Future Work

Appendices

A Manual Installation

B CAD Infrastructure Deployment

Bibliography

[A. Esteva 2017] B. Kuprel A. Esteva and R. Novoa. *Dermatologist-level classification of skin cancer with deep neural networks*, 2017. Available at https://www.nature.com/articles/nature21056. (Cited on page 5.)

[Cancer.Net 2020] Cancer.Net. *Skin cancer: Introduction*, 2020. Available at https://www.cancer.net/es/tipos-de-c. (Cited on pages 11 and 13.)

[Cancer.Net 2023] Cancer.Net. *Melanoma: Statistics*, 2023. Available at https://www.cancer.net/cancer-types/melanoma/statistics. (Cited on page 2.)

[Docker 2021] Docker. *Docker Documentation*, 2021. Available at https://www.docker.com. (Cited on page 75.)

[Farber 2020] Dana Farber. *Skin cancer, treatment*, 2020. Available at https://www.dana-farber.org/skin-cancer/. (Cited on page 15.)

[FastAPI 2021] FastAPI. *FastAPI Documentation*, 2021. Available at https://fastapi.tiangolo.com/. (Cited on page 75.)

[IBM 2020] IBM. *What is machine learning?*, 2020. Available at https://www.ibm.com/topics/machine-learning. (Cited on page 22.)

[IBM 2021] IBM. *Machine learning, explained*, 2021. Available at https://mitsloan.mit.edu/ideas-made-to-matter/machine-learning-explained. (Cited on page 22.)

[ISIC 2019] ISIC. *ISIC Challenge*, 2019. Available at https://challenge.isic-archive.com/. (Cited on page 3.)

[ISIC 2020] SIIM & ISIC. *SIIM-ISIC Melanoma Classification*, 2020. Available at https://www.kaggle.com/competitions/siim-isic-melanoma-classification/code. (Cited on pages 8 and 79.)

[ISIC 2022] ISIC. *The International Skin Imaging Collaboration*, 2022. Available at https://www.isic-archive.com/. (Cited on page 5.)

[J. Wang 2022] O. Shaik J. Wang R. Turko. *What is a Convolutional Neural Network?*, 2022. Available at https://poloclub.github.io/cnn-explainer/. (Cited on pages 6 and 9.)

[K. Stacke 2019] J. Unger K. Stacke G. Eilertsen and C. Lundström. *What is Weak AI*, 2019. Available at https://arxiv.org/pdf/1909.11575.pdf. (Cited on page 2.)

[Kaiming He 2013] Xiangyu Zhang Kaiming He and Jian Sun. *Deep Residual Learning for Image Recognition*, 2013. Available at https://arxiv.org/pdf/1512.03385v1.pdf. (Cited on page 46.)

[Korotkov 2012] K. Korotkov and R. García. *Computerized analysis of pigmented skin lesions: A review*, 2012. Available at https://www.sciencedirect.com/science/article/abs/pii/S0933365712001108. (Cited on page 5.)

[L. Davis 2019] S. Shalin L. Davis and A. Tackett. *Current state of melanoma diagnosis and treatment*, 2019. Available at https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6804807/pdf/kcbt-20-11-1640032.pdf. (Cited on page 1.)

[Leiter 2020] U. Leiter and T. Eigentler. *Epidemiology of Skin Cancer*, 2020. Available at https://link.springer.com/chapter/10.1007/978-1-4939-0437-2_7. (Cited on page 7.)

[Marghoob 2022] A. Marghoob and N. Jaimes. *Overview of dermoscopy*, 2022. Available at https://www.medilib.ir/uptodate/show/13521. (Cited on page 1.)

[Marr 2021] B. Marr. *What is Weak AI*, 2021. Available at https://bernardmarr.com/what-is-weak-narrow-ai-here-are-8-practical-examples. (Cited on page 2.)

[mrdbourke 2021] mrdbourke. *Learn PyTorch for Deep Learning: Zero to Mastery book*, 2021. Available at https://www.learnpytorch.io/. (Cited on page 75.)

[Partridge 2013] Dr. Partridge. *The Origins of Skin Cancer and How We Treat it Today*, 2013. Available at https://ohcare.com/the-origins-of-skin-cancer-and-how-we-treat-it-today/. (Cited on page 18.)

[Podman 2021] Podman. *Podman Documentation*, 2021. Available at https://podman.io/. (Cited on page 75.)

[Q. Ha 2020a] B. Liu Q. Ha and F. Liu. *Identifying Melanoma Images using EfficientNet Ensemble: Winning Solution to the SIIM-ISIC Melanoma Classification Challenge*, 2020. Available at https://arxiv.org/pdf/2010.05351.pdf. (Cited on pages 5 and 8.)

[Q. Ha 2020b] B. Liu Q. Ha and F. Liu. *SIIM-ISIC-Melanoma-Classification-1st-Place-Solution*, 2020. Available at https://github.com/haqishen/SIIM-ISIC-Melanoma-Classification-1st-Place-Solution. (Cited on page 8.)

[Russell 1950] Stuart J. Russell and Peter Norvig. *Artificial Intelligence A Modern Approach.*, 1950. Available at http://repo.darmajaya.ac.id/3800/1/Artificial%20Intelligence%20A%20Modern%20Approach%20%283rd%20Edition%29.pdf%20%28%20PDFDrive%20%29.pdf. (Cited on page 20.)

[Salary.com 2021a] Salary.com. *Hourly wage for Data Scientist*, 2021. Available at https://www.salary.com/research/salary/benchmark/data-scientist-i-hourly-wages. (Cited on page 70.)

[Salary.com 2021b] Salary.com. *Hourly wage for Data Scientist*, 2021. Available at https://www.salary.com/research/salary/posting/computer-and-information-research-scientist-salary. (Cited on page 70.)

[Salary.com 2021c] Salary.com. *Hourly wage for Data Scientist*, 2021. Available at https://www.salary.com/research/salary/listing/data-engineer-hourly-wages. (Cited on page 70.)

[Srivastava 2014] N. Srivastava. *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*, 2014. Available at https://jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf. (Cited on page 43.)

[sveltejs 2021] sveltejs. *SvelteKit Documentation*, 2021. Available at https://kit.svelte.dev/docs. (Cited on page 75.)

[Turing 1950] A. Turing. *COMPUTING MACHINERY AND INTELLIGENCE.*, 1950. Available at https://academic.oup.com/mind/article/LIX/236/433/986238. (Cited on page 19.)

[uslaev 2020] Eugene uslaev Alexande and Parinov. *Albumentations: Fast and Flexible Image Augmentations*, 2020. Available at https://www.mdpi.com/2078-2489/11/2/125. (Cited on page 9.)