Application of Artificial Intelligence Techniques to Biomedical Images:

AI and Digital Images Processing in Melanoma Detection

A. Giuliano Mirabella Galvín

Supervised by Luis Valencia Cabrera

Department of Computer Science and Artificial Intelligence,
Universidad de Sevilla

September 2020

Preface

The motivation behind this project is the interest in providing new knowledge and making an effort in the solution of healthcare problems on which biomedical engineering could really add a value, exploiting the benefits of conjugating healthcare and engineering skills.

Our previous project ideas

Among the different areas studied along the degree, the ones I have found especially powerful and promising are the AI and Digital Images Processing. Along my journey through the end of the studies, many ways of combining those worlds have come to my mind, so that when the time came to start the degree final project, I had a wide collection of potential ideas. To set a perspective the project should be considered from, I would like to describe some of those ideas my supervisor and I considered, but finally could not work on successfully.

- The initial idea, that led to the initial project assignment in the beginning of the grade, was to study the correlation between brain diseases and 3D brain images features. I had been working in topological analysis of nD images for a project in collaboration with the Applied Mathematics Department of the University o Seville, and I wanted to exploit that knowledge, together with more shape and distances-based features of the brain anatomical structures. The fact that prevented us from continuing through this path was the lack of a proper dataset, as well as the complexity of anatomical structures labeling, while being unable to contrast our progress with experts in the field.

- The second idea, that came in the beginning of 2020, was the application of AI techniques to results prediction in radiology treatment of lung cancer. My supervisor and I established contact with the entity "FISEVI" (Fundación Pública Andaluza para la Gestión de la Investigación en Salud en Sevilla) who was going to provide the dataset for the project. Unfortunately though, because of the *COVID-19 pandemic*, we faced many delays in obtaining the dataset from the institution, which ultimately made the execution of the project no longer possible, by June 2020.

In this context, as a second contingency plan after those significant efforts dashed by causes beyond our control, we made an analysis of more potential problems to solve, and, attending to the relevance and dataset availability we chose the "SIIM-ISIC Melanoma Classification" competition on Kaggle [1], whose goal is to detect Melanoma in skin lesion images. It is an emergent healthcare issue, as explained

later in introduction, and it offers the opportunity of applying topological features extraction, as well as many other digital images processing techniques.

About this document

I have decided to write this report about the project's life adopting a particular style, which I would like to introduce here.

As a student, I have had the opportunity -and the duty- of reading many books, often as compulsory handbooks for some subject, other times voluntarily in order to complement the skills developed along the degree. Now, there is, among all possible writing approaches I have met, one that has always been delighting to me, which is the style used in the book *Introduction to Electrodynamics*, by David J. Griffiths [2]. Quoting its preface:

"Unlike quantum mechanics or thermal physics (for example), there is a fairly general consensus with respect to the teaching of electrodynamics; the subjects to be included, and even their order of presentation, are not particularly controversial, and textbooks differ mainly in style and tone. My approach is perhaps less formal than most; I think this makes difficult ideas more interesting and accessible."

Now, I do not claim to be as good writer as David J. Griffiths, but I will try to approximate to its philosophy, in an humble intent of making you feel as comfortable as I felt when I was directly approached, questioned, and sometimes even challenged, while reading his masterpiece, what is considered to be the electrodynamics *bible*.

I really think this is the most beneficial, entertaining and enjoyable approach for the reader; may this decision not bother you, if you do not agree with me.

Without further ado, I hope you enjoy the project.

Abstract

The aim of this project is to create several intelligent systems capable of detecting melanoma in skin lesions images. Taking advantage of the “SIIM-ISIC Melanoma Classification" dataset on Kaggle, a hybrid Artificial Intelligence and Digital Images Processing based approach will be adopted to design images features extraction pipelines and procedures to correctly classify skin lesions into benign or malignant. For this purpose, a set of artificial networks based on different inputs will ultimately be developed, while conducting a deep analysis of every detail from the intuition behind the techniques used, through their implementation, up to the final performances comparison and discussion. The project's code is available in GitHub.

Resumen

El objetivo de este proyecto es el de crear varios sistemas inteligentes capaces de detectar melanomas en imágenes de lesiones de piel. Explotando el dataset de la competición “SIIM-ISIC Melanoma Classification" de Kaggle, se diseñarán, tomando un enfoque híbrido basado tanto en Inteligencia Artificial como en el Procesado de Imágenes Digitales, procedimientos para clasificar correctamente las lesiones de piel en benignas o malignas. Para este fin, se diseñará un conjunto de redes neuronales artificiales basadas en diferentes inputs, llevando a cabo un an álisis detallado de todas sus particularidades, desde la intuición básica detrás de cada téncnica usada, pasando por su implementación, hasta la comparación y discusión de los resultados finales obtenidos. El código que acompaña a este proyecto se encuentra disponible en GitHub.

Acknowledgments

I would like to express my sincere gratitude for all the support I was given by my project supervisor, Luis Valencia Cabrera. I am really thankful for his creativity, patience and encouragement I could not have successfully completed this project without.

A special acknowledgment goes to his colleague Miguel Ángel Martínez del Amor, whose effort in solving server-related problems has been crucial along the project development.

Last but not least, I vividly thank my family and friends, who have always been to me a constant and invaluable source of motivation.

Table of Contents

List of Figures

Chapter 1

Introduction

It is common knowledge the drastic benefits today's medicine experience when an engineering approach is adopted to solve clinical problems. An especially great effort has been spent in developing technologies capable of performing the diagnostic tasks.

In this project I have explored, as shown along this document, how far technology can go in automatic diagnostic. I will join both Digital Image Processing (DIP) and Artificial intelligence (AI) worlds' tools to make an intelligent system *learn* somehow the difference between a melanoma and a benign mole and then apply its knowledge to classify images into benign or malign skin lesion.

In this chapter an overview about the health problem to solve is provided, along with a general introduction to the proposed solution.

1.1 Skin Cancer

Skin Cancer is an issue. There seems to be a little disagreement about its prevalence: whereas some sources, such as the Skin Cancer Foundation (SCF), claim it to be the most prevalent type of cancer around the world [3, 4], others - such as the World Health Organization - place the non-melanoma skin cancer *only* in the 5^{th} position, and the melanoma in 19^{th}.

Whichever statistic may be right, we can safely assume the skin cancer to be one of the most important issue to challenge in today's medicine. According to SCF, here are some stats & facts [3] to strengthen this hypothesis:

- "more people are diagnosed with skin cancer each year in the U.S. than all other cancers combined"
- "at least one in five Americans will develop skin cancer by the age of 70"
- "in the past decade (2010 – 2020), the number of new invasive melanoma cases diagnosed annually increased by 47 percent"

Unfortunately, even though the above sentences only considered American population, there is no reason to feel left out of these statistics, as according to the American Institute for Cancer Research [5], USA is *only* in the 17^{th} position in Countries with highest rates of melanoma list.

Skin cancer can be classified into three general types: basal-cell carcinoma, squamous-cell carcinoma and melanoma. However, a simpler classification is generally used and the first two skin cancers are commonly referred to as non-melanoma skin cancers since, as it can be read below, melanoma deserves a standalone concept due to its aggressiveness.

1.2 Melanoma

Melanoma is the skin cancer that develops in the cells responsible for melanin production (melanocytes) and, despite being the least common type, it is by far the biggest threat, causing $\sim 75\%$ of skin cancer deaths.

As it occurs with other cancers, early and accurate detection are crucial to increase the effectiveness of the treatment. According to SCF, when detected early enough, the 5-year survival rate for melanoma is 99 percent, which should be a strong incentive to develop melanoma detection technologies, such the one I am trying to design in this project.

1.3 The Dataset

The dataset this project works on comes from the "SIIM-ISIC Melanoma Classification" competition [1], a public competition in Kaggle.

Kaggle [6] is an online community of data scientists and machine learning practitioners. As described in the citation above, "it allows users to find and publish data sets, explore and build models in a web-based data-science environment, work with other data scientists and machine learning engineers, and enter competitions to solve data science challenges".

The Society for Imaging Informatics in Medicine (SIIM) is, textually from [7], "the leading healthcare organization for informatics in medical imaging. Its mission is to advance medical imaging informatics through education, research, and innovation in a multi-disciplinary community".

As Kaggle's competition keep explaining, for this dataset, "SIIM is joined by the International Skin Imaging Collaboration (ISIC) [8], an international effort to improve melanoma diagnosis. The ISIC Archive contains the largest publicly available collection of quality-controlled dermoscopic images of skin lesions."

Thanks to Kaggle's competitions and their rewards, participants are challenged to develop as-good-as-possible solutions for the problem under consideration, and while they compete, work and learn, science happen to become step by step more powerful as new tools are continually and globally forged. And this all is especially beneficial when the problem to solve is a disease diagnostic or treatment optimization, probably from the biased view of a health engineer but also as a human being.

1.4 About the Project

In this section the purpose of the project will be presented, highlighting the main areas it is going to be based on to provide a solution to the healthcare problem described above: Artificial Intelligence (AI) and Digital Images Processing (DIP).

1.4.1 The Project Goal

The ultimate goal of this project is to experiment with the application of AI techniques both to original images and numerical features, extracted by means of Digital Images Processing methods, to predict if a skin lesion image is benign or malignant. Besides that, the performance of each experiment, technique, and developed system will also be evaluated and discussed.

1.4.2 A Little Preface about DIP Techniques

This project is an experiment, that should be taken as a first humble attempt to help solving the problems previously described, deepening into the study of some existing AI and DIP techniques, with some experimental approach combining different skills. The fundamental idea, again, is to apply AI techniques both on the original images and on certain images topological or numerical features extracted by means of digital image processing methods, and then to compare the results obtained, in order to extract some conclusions about the relevance of the different models and features. One important detail to take into account for the latter though is that all along the project, despite the interesting questions and thoughts emerging at different points of the experiments and comparisons of metrics, I will have no final, definite clue about how accurate the results of those numerical features extracted from images are. Let me explain it better below.

When implementing new algorithms for solving a medical DIP problem, the globally accepted evaluation method is to compare the output of the considered process with the output of an analogue one hand-made by a doctor; unfortunately, the dataset does not provide any other kind of information about the output of any process that may be applied to images, so I have no chance to test if what I am doing is right or not.

What I am trying to point here is that the proposed inner DIP techniques are not valid as long as somebody prove they are, and I can't. I can only evaluate how well my system can correlate those extracted features with the benign or malignant nature of the skin lesion, but not *how accurate those features are*, until an expert in the specific medical domain can validate these achievements.

In this project, I will hopefully provide a general understanding about the fundamentals of both used technologies (chapter 2), a careful description on how the images features are going to be extracted and how the AI systems are designed (chapter 3); then the results and discussion will be exposed (chapter 4), and finally the main conclusions drawn from this project will be outlined, along with some possible lines of future work that might be conducted (chapter 5).

I would like to end this introduction quoting a Kaggle's competition paragraph:

"Melanoma is a deadly disease, but when caught early, it can be cured almost always with minor surgery. Image analysis tools that automate the diagnosis of melanoma will improve dermatologists' diagnostic accuracy, which has the opportunity to positively impact millions of people."

Let's see then what we can humbly do to positively impact millions of people.

Chapter 2

Overview of the technologies involved

The aim of this chapter is to introduce those readers who may not be familiar with this branch of engineering to the fundamentals of both technologies involved along the project: Digital Images Processing (DIP) and Artificial Intelligence (AI). Additionally, some fundamentals will be provided as a background for following chapters.

2.1 Digital Images Processing

DIP is the use of computer science algorithms to treat, modify, transform, and process digital images. It can be thought of as a subcategory of digital signal processing where the signal has two important properties:

- it is usually 2 or 3-dimensional (2D or 3D-images, or 2D-videos), and
- can be grayscaled or colored

In a mathematical approach, a more formal definition of a digital image is a function f:

$$f : \mathbb{N}^n \to \mathbb{N}^c$$

Where n is the dimension of the image and

$$c = \begin{cases} 1 & \text{if the image is grayscaled} \\ 3 & \text{if the image is colored} \end{cases}$$

In this project we will analyze 2D-colored images, something that, can be thought of, at least by now, 2D-matrices containing a 3D-vectors in each node expressing the red, green, and blue value (RGB) the node takes in real world.

Note that the range of an image function is above represented as $\mathbb{N}^c$ for simplicity, but is not actually always that; by a simple processing of every pixel value, it can be rewritten as $\mathbb{R}^c$ or similar. In fact, in our project the range will be normalized to $[0, 1]^c$.

DIP and Computation. This project code will be entirely written in Python language, so it is worth explaining how images have been handled in our case. Thanks to *numpy*, a package for tensors treatment, the management of matrices is made very simple and intuitive, as if they were scalar numbers. Besides that, the most used package in the project will be *scikit-image*, a module with plenty of useful image processing built-in functions.

Below, I am going to formally define some of the simplest operations that can be executed on digital images, in order to strengthen the idea of the matrix nature of images. When I assume the image is grayscaled, I do it for the sake of simplicity, but for RGB images the corresponding functions would be equivalent.

2.1.1 Values and Size editing

Consider a grayscaled $(a \times b \times ... \times z)$-sized nD image X on a domain set $P \in \mathbb{N}$ to a range set Q whose minimum and maximum values are: q_{min} and q_{max}, respectively:

$$X : P^n \to Q$$

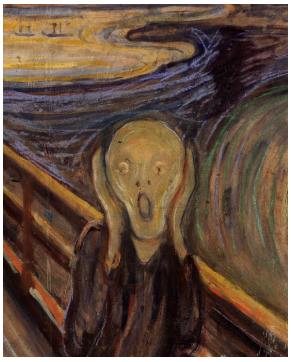
A simple gray-values normalization would produce a new image $Z = z_{ij..k}$:

$$z_{ij...k} = \frac{x_{ij...k} - q_{min}}{q_{max} - q_{min}}$$

A crop of the image (figure 2.1) into a new $((a', a'') \times (b', b'') \times ... \times (z', z''))$-sized one would be a new image $Z = z_{ij..k}$:

$$z_{ij...k} = \begin{cases} x_{ij...k} & \text{if a'<i<a", b'<j<b", ..., z'<k<z"} \\ 0 & \text{otherwise} \end{cases}$$

(a) Original image

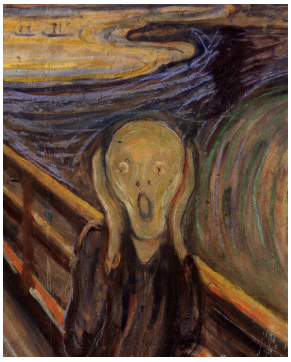


(b) Cropped image

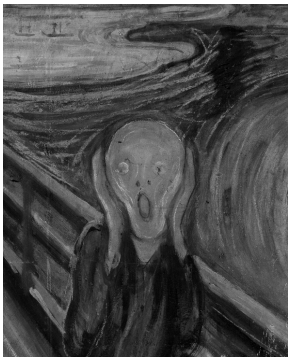
Figure 2.1: Cropping of an image, as a matrix borders suppression.

As explained above, an RGB image is a matrix, just as a grayscaled, but where each element has three components (is a vector); so how is an RGB image $X = x_{ij..k}$ converted to a grayscaled $Z = z_{ij..k}$ (figure 2.2)?

$$z_{ij...k} = 0.3x^R_{ij...k} + 0.59x^G_{ij...k} + 0.11x^B_{ij...k}$$



(a) Original image



(b) Grayscaled image

Figure 2.2: Converting an RGB image to grayscale.

being $x^y_{ij...k}$ the y color component of the $ij...k$ element. A curious fact is that a grayscaled image *is not the mean* of the three color components; because of the human eye different sensibility to red, green and blue, a mean value image would look much darker than the grayscaled images we are used to [9].

2.1.2 Filtering

The filtering of a digital image is obtained by the convolution of a kernel over it. A convolution is an operator that, given an $(n \times m)$ image X and a $(a \times b)$ kernel f, returns a matrix formed by the sum of all elements obtained by the element-wise multiplication between the kernel and certain possible $(a \times b)$ "windows" of X.

The formal definition of a convolution of a kernel f over a 2D image X is given by (from [10]):

$$z_{i,j} = \sum_k \sum_l x_{k,l} f_{k-i,l-j}$$

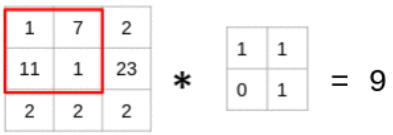
Let's visualize it with an example, taken from [11]. Let the image to convolve be:

$$X = \begin{pmatrix} 1 & 7 & 2 \\ 11 & 1 & 23 \\ 2 & 2 & 2 \end{pmatrix}$$

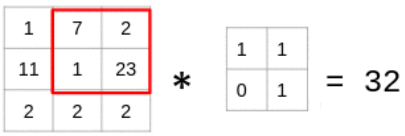
And the kernel:

$$f = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$$

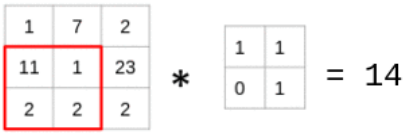
Graphically, the process of convolving is (recall this is an element-wise multiplication followed by the corresponding addition):



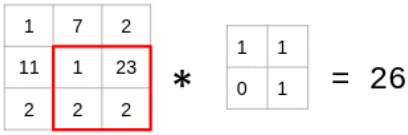
(a) 1^{st} window



(b) 2^{nd} window



(c) 3^{rd} window



(d) 4^{th} window

and the convolution Z is a matrix formed by the result of every step:

$$Z = \begin{pmatrix} 9 & 32 \\ 14 & 26 \end{pmatrix}$$

There are plenty of useful convolution kernels, such as the mean, Gaussian or Laplacian ones, and they can be of any size:

$$\frac{1}{4}\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$$

a)

$$\frac{1}{9}\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

b)

$$\begin{pmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{pmatrix}$$

c)

$$\frac{1}{273}\begin{pmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{pmatrix}$$

d)

Figure 2.4: a)2x2 mean filter. b)3x3 mean filter. c)3x3 laplacian filter. d)5x5 gaussian filter.

See [12] for more info about digital filters.

2.1.3 Morphological Operations

Morphological operations are also based on convolutions. This time though, the filter is commonly known as "structuring element" and it has an origin. Intuitively, the goal of a morphological function is to detect those pixels fulfilling a certain condition on its neighboring pixels, where such neighborhood is defined by the structuring element.

There are four main morphological functions: *erosion*, *dilatation*, *opening* and *closing*, [13].

Erosion. The erosion of a binary image I by a structuring element F is given by:

$$I \ominus F = \bigcap_{f \in F} I_{-f}$$

Where X_y is the translation of X by the vector y:

$$X_y = \{x + y \mid x \in X\}$$

Let's see an example. Given the structuring element F:

$$F = \begin{pmatrix} \times & \times & \times \\ \times & O & \times \\ \times & \times & \times \end{pmatrix}$$

being O the origin, and a binary image I, the erosion operation acts as shown in figure 2.5.

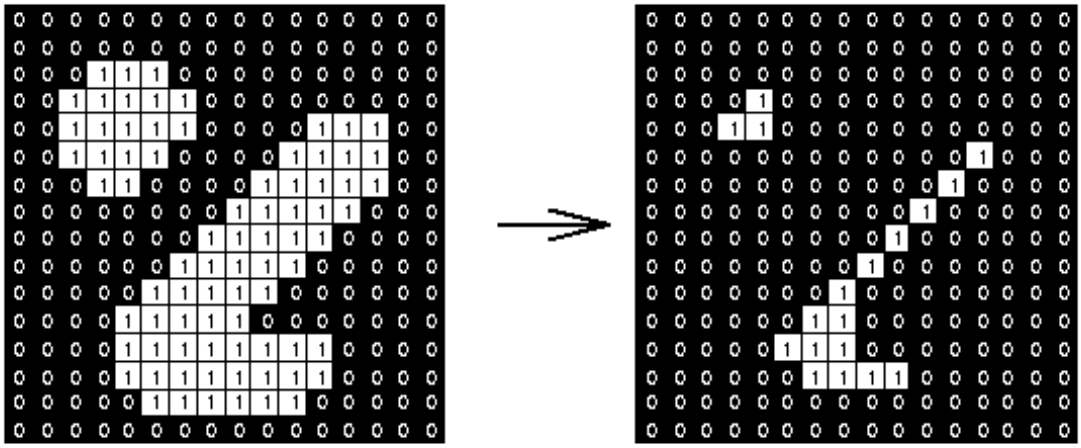


Figure 2.5: erosion of an image by a 3x3 isotropic structuring element

It can be thought of as the intersection of all matrices $I - f$ where $f \in \{(-1,1)$, $(0,1)$, $(1,1)$, $(1,0)$, $(1,-1)$, $(0,-1)$, $(-1,-1)$, $(-1,0)$, $(0,0)\}$, or, in a more friendly way, as the "switching off" of all pixels not fulfilling *all* neighborhood conditions expressed by F relatively to the origin, *i.e.* not having *all of* the neighbors expressed in F.

Dilatation. The dilatation of a binary image I by a structuring element F is given by:

$$I \oplus F = \bigcup_{f \in F} I_f$$

It is illustrated by figure 2.6.

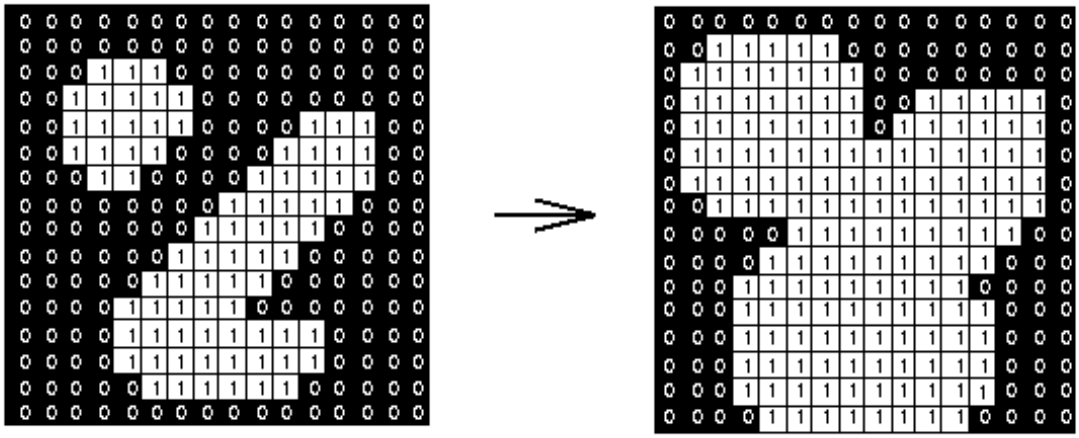


Figure 2.6: dilatation of an image by a 3x3 isotropic structuring element

Again, it can be thought of as the union of all matrices $I + f$ where $f \in \{(-1,1), (0,1), (1,1), (1,0), (1,-1), (0,-1), (-1,-1), (-1,0), (0,0)\}$, or, in a more friendly way, as the "switching on" of all pixels fulfilling *at least one* neighborhood condition expressed by F relatively to the origin, *i.e.* having *at least one of* the neighbors expressed in F.

Opening. The opening of a binary image I by a structuring element F is given by:

$$I \circ F = (I \ominus F) \oplus F$$

i.e. it is the dilatation of the erosion. It is used to suppressing small-enough dark components, and it can join white regions that weren't connected.

Closing. The closing of a binary image I by a structuring element F is given by:

$$I \bullet F = (I \oplus F) \ominus F$$

i.e. it is the erosion of the dilatation, used to suppressing small-enough white components instead. Opening and closing are idempotent operations and among their many applications there are the following ones.

Salt-and-pepper noise correction: salt-and-pepper noise is a form of noise sometimes seen on images that present sparse white and black pixels in black and white regions, respectively. When the image undergo a closing, the erosion will hopefully make the undesired black components (pepper) disappear, and not grow back with the dilatation. Likewise, opening will make white small components in black regions disappear; in figure 2.7, from [14], an example is shown.

(a) salt noise

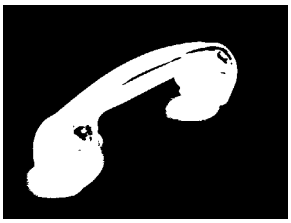
(b) Opened image

(c) pepper noise

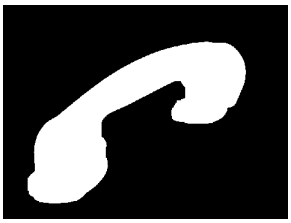
(d) closed image

Figure 2.7: salt and pepper noise correction

Irrelevant information suppressing: same as above, when the structuring element is large enough.



(a) original image



(b) closed image

Figure 2.8: Irrelevant information suppressing

In the following section, likewise the DIP, the AI world's fundamentals will be shown.

2.2 Artificial Intelligence

AI is the branch of computer science that study how to design systems that simulate what us humans understand as intelligence. The concept of intelligence is probably one of the most unknown things in our universe, so it is easy to imagine how complex the task of simulating it in a deterministic machine is. To be more specific, the main AI sub-areas I am going to use is Machine learning, including also Deep learning.

2.2.1 Machine and Deep Learning

Machine learning consists of "algorithms that parse data, learn from that data, and then apply what they have learned to make informed decisions" [15].

Among others, machine learning studies and designs systems to correctly predict certain variables, while also being capable of comparing their predictions with the expected values of such variable and auto-improving themselves.

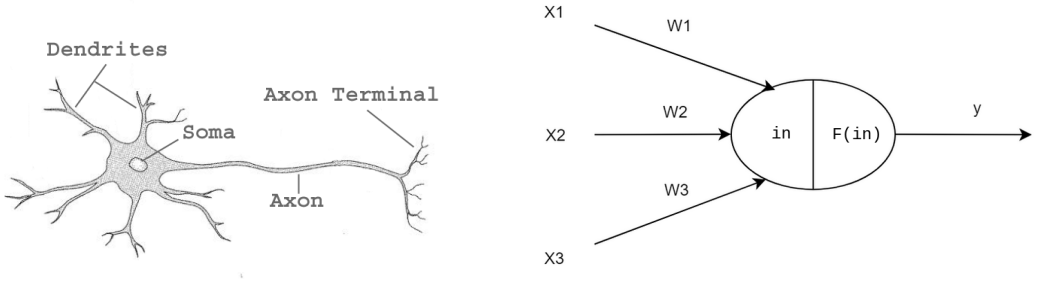
Deep learning is an evolution of Machine learning that focuses on systems that have a considerable amount of hidden work and information. Among the strongest tools of machine and deep learning we have Artificial Neural Networks (ANN). An ANN is a human-brain-inspired system formed by a set of artificial neurons-inspired units called perceptrons; we will use it in two flavors: Multi-Layer Perceptron (MLP) and Convolutional Neural Network (CNN). Both are machine learning's techniques, while in general only the CNN has deep enough topology and hidden enough information to be considered a deep learning tool.

2.2.2 The Perceptron

The basic structural unit of ANN is an artificial neuron or a perceptron. Perceptrons interconnect with each other receiving, processing and emitting signals to others, thus finally forming the network.

A neuron is an electrically excitable cell that takes up, processes and transmits information through electrical and chemical signals. A typical neuron is divided into three parts: the dendrites, where signals from others neurons are received, the cell body (what contains the soma in figure 2.9), where the signal is processed and redirected, and the axon, that emits the stimulation to others neurons' dendrites through structures named synapses [16].

In the following figure a real vs. artificial neuron is shown:



(a) Our neurons

(b) Artificial neuron

Figure 2.9: real neurons vs. artificial ones

Inspired by the real one, the artificial neuron has the following *connections path*:

- *inputs* from other neurons $(x_1, x_2, \dots)$,
- *weights* $(w_1, w_2, \dots)$, in order to balance the relative influence of each input to the neuron, to forming the final input $in = \sum_{j=1}^{n} w_{ij} x_j$,
- a *processing* F of the input, and
- an *output* y (that can be sent to another neuron).

Now, the perceptron output behavior can be modified as one likes by altering F. However, it is generally thought of - just as a real neuron - as a dichotomous phenomenon, *i.e.*, the perceptron can activate, emitting an output $y \neq 0$, or remain inactive, emitting $y = 0$. This behavior is fulfilled by:

- $x_0 = -1$, that works as an activation threshold
- F, generally named "activation function", being frequently one of the following:

$$sgn(in) = \begin{cases} 1 & \text{if in} > 0 \\ -1 & \text{if in} \leq 0 \end{cases}$$

$$threshold(in) = \begin{cases} 1 & \text{if in} > 0 \\ 0 & \text{if in} \leq 0 \end{cases}$$

$$relu(in) = \begin{cases} in & \text{if in} > 0 \\ 0 & \text{if in} \leq 0 \end{cases}$$

As a result of those functions, the perceptron activates when the weighted sum of $x_1, x_2, ...$ plus $x_0 = -1$ is grater than 0, *i.e.* when the weighted sum of $x_1, x_2, ...$ is grater than 1. However, the perceptron activation can also adopt a continuous response to the input, with the activation function being for instance the sigmoid function:

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

To build a complete network, perceptrons join together in layers, and layers join to form a Multi-Layer Perceptron (MLP).

2.2.3 MLP

An MLP is a mathematical model based on a graph structure whose nodes are artificial neurons (figure 2.10, from [17]).

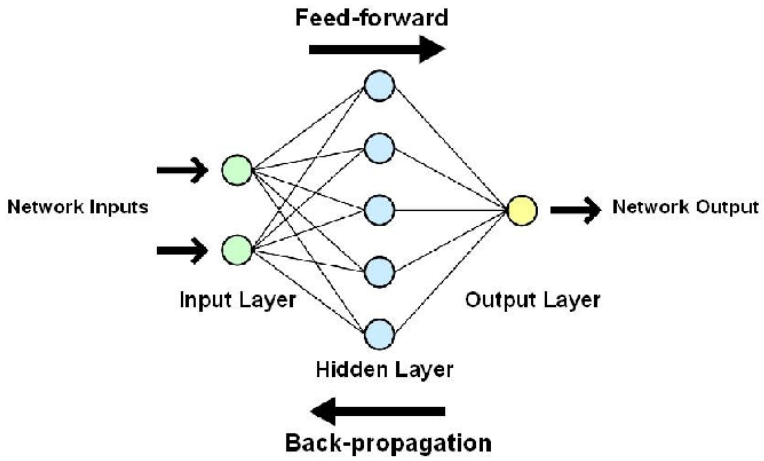


Figure 2.10: A simple MLP

Every node is connected to others through directed and weighted edges, modeling the axon to dendrites connection. Each edge $i \to j$ propagates the output of the perceptron i to the j and has a weight w_{ij} that determine the strength of the connection $i \to j$, which simulates the synapse. Each node computes its output based on the inputs it receives and works likewise as an input for others nodes [18].

As you can see in figure 2.10, similarly to a single perceptron, an MLP has an *input layer*, which accepts the input parameters, an *output layer* which emits the output and some *hidden layers*, which perform an inner processing. An MLP with n nodes in the input layer and m nodes in the output layer behaves just like an $\mathbb{R}^n \to \mathbb{R}^m$ function, so that it can be used as a classifier in $\mathbb{R}^n$ into m classes.

So, given this graph-based structure, how does an MLP learn? Here is a question: when you were a kid, did anyone ever had to teach you the definition of what is a dog and what a cat? The answer is - or at least should be - no. You just *learned* it by listening the word "dog" whenever any information of your surrounding stimuli referred to a dog, and the same with cats. If that stimulus is repeated enough times, a two-years-old kid's brain will eventually relate a dog with the word "dog", a cat with the word "cat" and so on, and yet we call it *learning* even without any zoological formal definition.

The process of training of an ANN follows a dataset split into *training* and *test*, and consists in performing a similar reward-punishment formula over certain inputs whose corresponding output, given by certain labels, are already known. For this purpose, the network shall:

- predict labels on *training set*,
- be given a *punishment* if outputs are incorrect

Once the *training* is done, the network performance can be tested on the *test set*. During the *training*, the network will initially speculate rather than predict, since it has no clue about what the relation between inputs and outputs is supposed to be. It is not until the network is trained that the first reasonably correct results come out.

If you think of it, the baby from the analogy experiences a similar "reward-and-punishment" process; in fact, when he/she points a cat with his/her finger and says "dog", the adult who is looking after him/her should hopefully correct the *prediction* by simply saying "not a cat, it's a dog!". The MLP are just oversimplified versions of our brains, and they need to *learn* the same way.

But how do we reward or punish our network? You may have noticed there is another flow of information in figure 2.10, apart from the forward communication path explained above: the back-propagation.

The back-propagation is the backward flow that accomplish the task of making the network learn by updating the *weights* of every connection once the punishment comes from the output layer. The algorithm is built upon the gradient descent idea: it simply drags the error result of comparing the prediction with the known label on the output layer backward to the hidden layers up to the input one, lowering or increasing (depending on the sign of the error) the weights of the connections that activated for that wrong prediction. When the prediction is correct the error in the output layer is zero, so that no weights update is dragged to the inner layers.

There are some particular parameters, called hyper-parameters, that influence the back-propagation, thus intervening in an ANN training. Especially, the following have been studied for this project:

- Batch size: is the number of inputs the systems accepts before a new updating of its weights is done (the back-propagation algorithm is invoked). The higher this number is, the *slower* the network will learn. It may sound contradictory though, why would anyone want a system to learn slowly? Don't worry, we will come back to that in section 3.6.1.

- Epochs: this number indicates how many times the system is going to accept the complete dataset, *i.e.* how many times each input case is going to be visited from the network. The higher the number of epochs is, the more the system will happen to memorize input cases. Again, it is a little counter-intuitive now, but *memorizing* is not as good as it sounds.

2.2.4 CNN

Could an MLP accept an image, which is a 2-dimensional matrix, as an input? The answer is yes. Pixel values could be flattened to a 1-dimensional vector and can be processed just like any other input; but there is actually a better way to do it. During the flattening process, an important property is lost: the spatial position. Thus, a more powerful system, with structures capable of accepting a more-dimensional input, is needed to solve properly this problem. Such system is called CNN and such structures are called filters.

Filters (or kernels) are the building blocks of CNNs. Kernels are used to extract the relevant different-level features from the input image using the convolution operation, described in 2.1.2. Filters are capable of extracting important properties, such as edges and geometrical structures, in both high or low level of cognition.

As shown in the following figure taken from [19], convolving an image results in a feature map.

Figure 2.11: Output of convolution

and the incredible power of CNNs comes once you understand those features are actually learned and extracted automatically. Just like MLP learned by updating its weights on connections, a CNN learn by updating its kernels elements, so there is no need to explicitly define them. When a punishment is given to a prediction, every connection that led to that prediction is updated; when instead the prediction is correct, weights rest the same (figure 2.12).

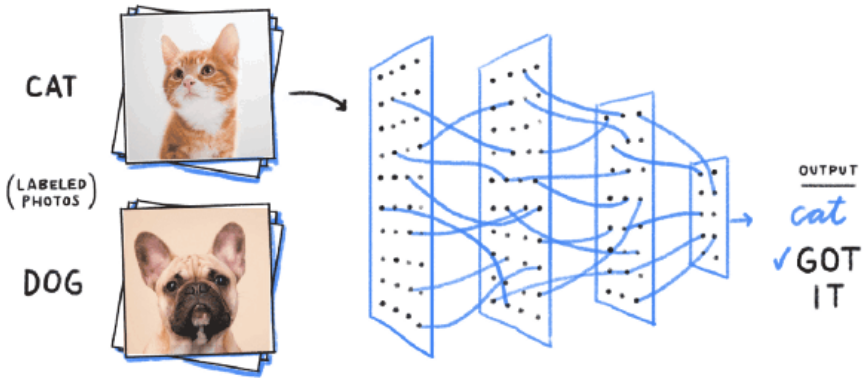


Figure 2.12: A correct prediction does not update the weights of the paths activated

Therefore a continuous updating of kernels weights is carried out, which ultimately leads to a more accurate and relevant features extraction.

So far, a general introduction to DIP and AI worlds has been provided, from the mathematics fundamentals, through their simple intuition as a real-world inspired tech tool, up to several of their techniques that are going to be applied along the project. Now, it is time to start discussing what the methods of the project are and how ultimately the melanoma detector system is going to be designed.

Chapter 3

Methods

So the melanoma is threatening, and we have plenty of images and some engineering tools. What to do now? How can the information in a 2D matrix help to distinguish malignant melanoma from benign?

In this chapter I will carefully describe the recipe for the proposed solution, starting by a general exploration of the project dataset, then providing an outline of the path the images describe through the project, and finally focusing deeply on how each process works.

3.1 Data Exploration

To start with, we have to do what in data science is named an *exploratory data analysis* (EDA). An EDA is used to summarize and visualize the dataset's main characteristics, which could lead to new data collection and systems architecture ideas.

The columns The dataset is composed by images in DICOM and JPEG formats, and a .csv file, whose columns are:

- `image_name`: unique identifier, points to filename of related DICOM image
- `patient_id`: unique patient identifier
- `sex`: the sex of the patient
- `age_approx`: approximate patient age at time of imaging
- `anatom_site_general_challenge`: location of imaged site
- `diagnosis`: detailed diagnosis information (train only)
- `benign_malignant`: indicator of malignancy of imaged lesion (just a verbal expression for the target column, (0 is benign and 1 is malignant))
- `target`: binarized version of the target variable

Let's visualize the *.csv* file info:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 33126 entries, 0 to 33125
Data columns (total 8 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   image_name                    33126 non-null  object
 1   patient_id                    33126 non-null  object
 2   sex                           33061 non-null  object
 3   age_approx                    33058 non-null  float64
 4   anatom_site_general_challenge 32599 non-null  object
 5   diagnosis                     33126 non-null  object
 6   benign_malignant              33126 non-null  object
 7   target                        33126 non-null  int64
dtypes: float64(1), int64(1), object(6)
memory usage: 2.0+ MB
```

Figure 3.1: dataframe general info.

So there are 33126 rows and 8 columns, as expected. Furthermore, we can also observe there are some missing values:

column	missing proportion
anatom_site_general_challenge	0.015909
age_approx	0.002053
sex	0.001962

Their distribution Inspired by some very interesting Kaggle Notebooks [20], let's see how the columns values are distributed. The target variable:

	target	occurrences	proportion
0	0	32542	98.237034
1	1	584	1.762966

It's easy to see how unbalanced it is, *i.e.*, how different the number of cases belonging to each class are present:

Figure 3.2: Target distribution.

Now let's visualize the others variables distribution with respect to the target. Starting with sex:

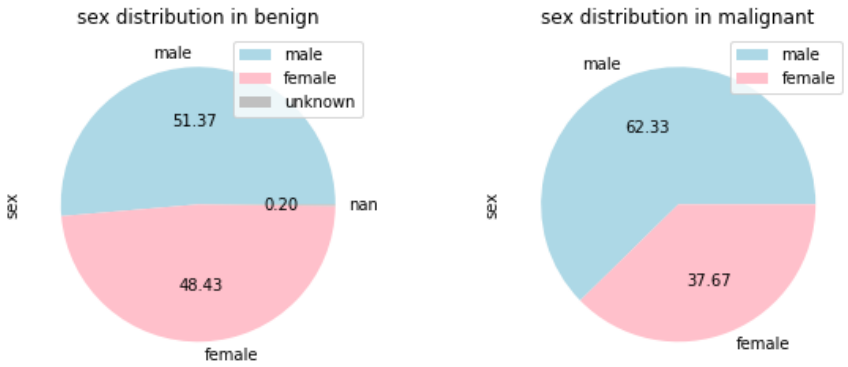


Figure 3.3: Sex distribution vs. target.

It seems to be a little bit unbalanced. All the 0.2% cases whose gender is unknown are benign, but it's no surprise since only 1.7% of the entire population is malignant.

The age distribution is easier to visualize in a density histogram:

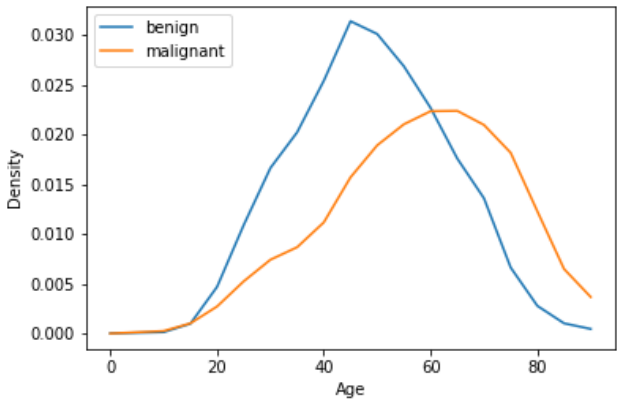


Figure 3.4: Age distribution vs. target.

Again, the mean is a little bit displaced to the right. Now it's the turn for the *anatom_site*:

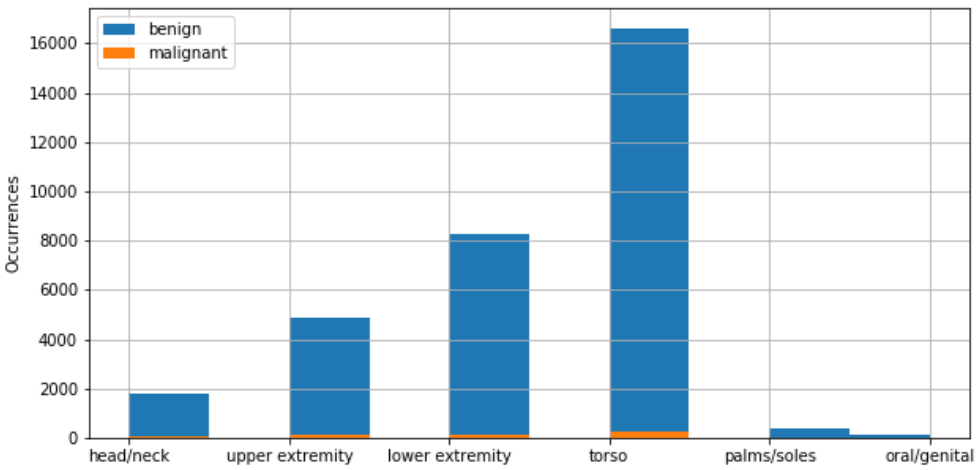


Figure 3.5: Sex distribution vs. target.

And the *diagnosis*:

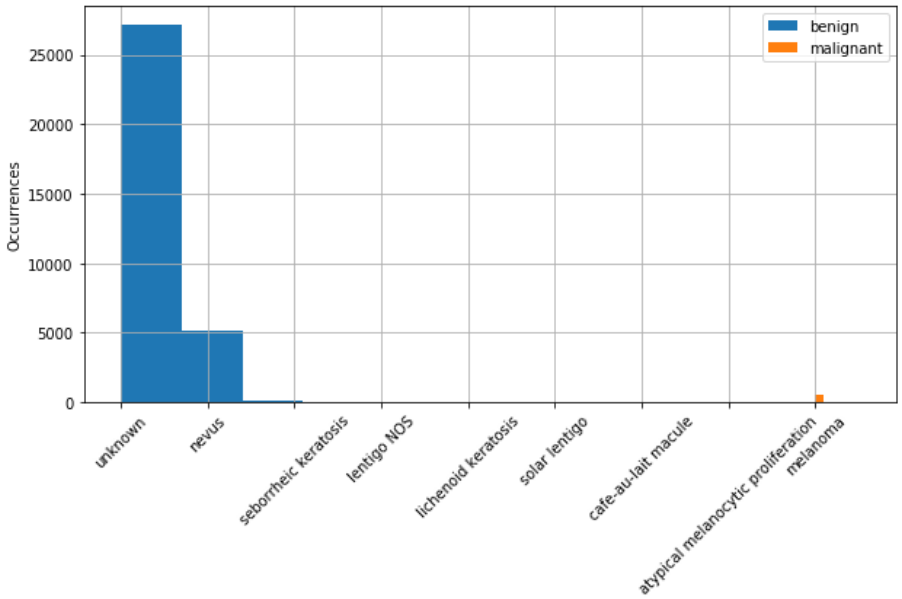


Figure 3.6: Diagnosis distribution vs. target.

Of course, melanoma value for the diagnosis column has the same meaning of 1 in target as all melanoma are malignant and all non-melanoma benign.

Let's take a look now at the *.dcm* files. DICOM (Digital Imaging and Communication On Medicine) is an images and data transmission standard in medicine. A DICOM file consists of a header and image data sets packed into a single file.

Let's see what the DICOM header is like:

	image_name	dcm_modality	dcm_study_date	dcm_age	dcm_sex	dcm_body_part_examined	dcm_patient_orientation	dcm_photometric_interpretation
0	ISIC_3918318	"XC"	20200519	065Y	M	TORSO		YBR_FULL_422
0	ISIC_4975848	"XC"	20200520	070Y	M	LOWER EXTREMITY		YBR_FULL_422
0	ISIC_8647334	"XC"	20200519	045Y	F	LOWER EXTREMITY		YBR_FULL_422
0	ISIC_2887033	"XC"	20200519	050Y	F	UPPER EXTREMITY		YBR_FULL_422
0	ISIC_8690825	"XC"	20200519	040Y	F	UPPER EXTREMITY		YBR_FULL_422

Figure 3.7: DICOM header

The *dcm_photometric_interpretation* parameter is clarifying that images are interpreted by default in a YBR color space, so a conversion to RGB must be done before visualization. To finish with, let's visualize some random images, half benign and half malignant:

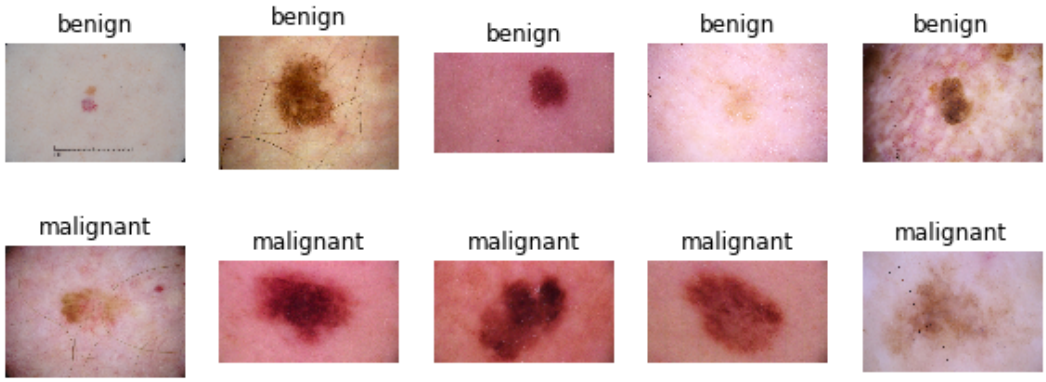


Figure 3.8: Some random images.

From our unfamiliarity with the problem domain, can we observe any difference between benign and malignant images? Maybe the shape, which seems to be more irregular in malignant... however, let there be no bringing forward of any guess by now, we will deepen more in following sections.

3.1.1 Dataset Strategy

After the EDA, some conclusions can be extracted to lead the project into an optimization of all the dataset information provided.

Effective columns There are some dependent columns which implicitly means the same as target, which should be obviously not used in training, as the goal of a predictor is to predict unknown information. These are the columns *benign_malignant* and *diagnosis*.

Class Imbalance Correction We have seen how strongly unbalanced the target variable is. Only 1.63% of the cases are malignant, which is an unacceptable dataset lack of equilibrium to train any intelligent system on.

Furthermore, recall this is a hybrid project, made on the purpose of joining the DIP and AI worlds together with an experimental approach. The images processing along the pipeline are not quite computationally cheap, so, the less images we consider, the less time we are going to spend in image outputs computation.

The solution we propose to this problem is a down-sampling mixed with class-balance factor.

- Down-sampling: the dataset is going to be reduced to randomly chosen 5000 cases plus the exceeding malignant ones, totaling 5509 cases, being 4925 benign and 584 malignant, 89.4% and 10.6%, respectively.
- Class-balance: once down-sampled the dataset, the AI techniques are going to be applied separately to:
  - 1168 cases, of which 50% will be malignant and 50% benign
  - the whole 5509 set cases, with a class-balance correction factor

Later, in section 3.6, both those strategies will be carefully explained, as well as which of them each experiment adopts.

3.2 The Digital Images Processing Pipeline

Now, it is time to state what the flow of information is from the initial pixels values of the starting images dataset to the final dataframes obtained, *i.e.* what processes the images are going to experience. The general pipeline diagram is shown in figure 3.9. The acronyms meaning are listed below.

- HR: hairs removal process; it removes hairs structures from the original images.
- SEG: segmentation process; it identifies which pixels of the image belong to the skin lesion and which don't.
- ECE: Euler characteristics extraction.
- SFE: statistical features extraction, from RGB original images, using the segmented ones.
- CNN: Convolutional Neural Network system.
- MLP: Multi-Layer Perceptron system.

So images are going to be treated by a hairs removal step and topologically analyzed, then segmented and shape and color analyzed, and finally all this data is going to be accepted by MLPs and CNN systems. Each of these processes is going to be carefully described in below sections, where a general intuition as well as a formal definition will be presented.

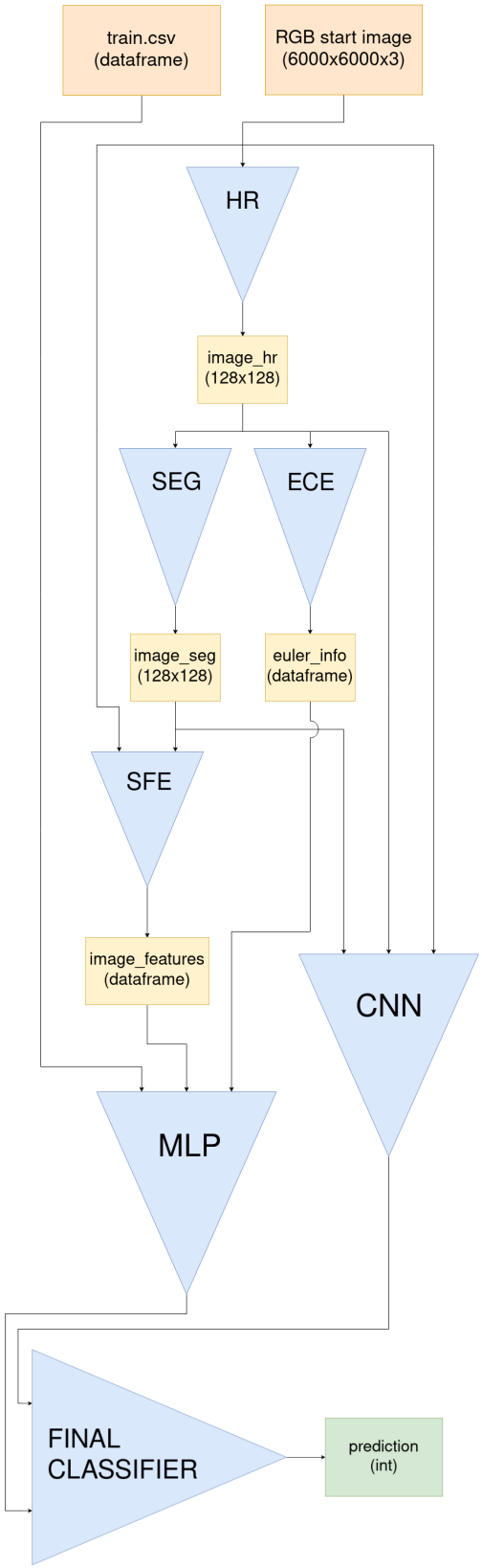


Figure 3.9: DIP Pipeline.

3.3 Segmentation

The segmentation process (SEG) provides the following conversion:

- Input: an RGB (6000x6000) image
- Output: a segmented (128x128) image

The segmentation of an object X in an image consists in a binary classification of each pixel into those belonging to X (foreground) and those not (background). Its aim is to identify the object of interest in the image, to discern which pixels it is made of. For example, here is the segmentation of a man taking a photograph with his camera:

(a) Original Image

(b) Segmented image

Figure 3.10: Segmentation of a man with his camera.

To produce the final binary image (b), each pixel in the original image (a) has been classified to be a man's pixel (white) or a background's pixel (black). At first sight, the segmentation output seems to quite fulfill the discrimination. However, there are still regions considered to be part of the man without being so (those far buildings for instance) and vice versa, such as the shirt leaning out his neck, and his face too!

This occurs because of the method used to segment that image, which in this case happens to be a threshold function assigning the value 1 to those pixels with higher values than a threshold t and 0 to others. Despite its apparently simplicity, it is trickier than it seems: the threshold t must be carefully chosen to optimize the segmentation performance; in this case, which applies the Otsu's method [21], such threshold is chosen to minimize the intra-class variance, defined as a weighted sum of variances of the two classes:

$$\sigma_w^2(t) = w_0(t)\sigma_0^2(t) + w_1(t)\sigma_1^2(t)$$

being 1 and 0 the classes, σ_i^2 the i class variance, and w_i the i class probability, computed in turn from the image histogram.

What I am trying to illustrate here by pointing out the quite low performance of the process, as well as its apparently hidden sort of mathematical complexity, is not how bad the algorithm is, but instead how non-trivial it is to design DIP algorithms and ideas to make a computer system automatically identify objects in images. We have spent decades in thinking of new solutions and still don't have a

general magic one. Instead, depending on the problem, which in real life turns to be by far harder than the toy image above (an organ segmentation in a 3D magnetic resonance image, for instance), an algorithm should be preferred to another. AI has joined DIP and become nowadays our best chance to accomplish this task, making computer vision applications each day better and better at recognizing objects in images and videos.

To me, it is really intriguing how far DIP algorithms alone are from accomplishing this task and how computationally expensive it is, whilst us humans can recognize our mom's face in about 10^{-1} seconds. We are teaching machines to recognize objects, but either machines are very different from what we thought, or we are really bad teachers.

So, coming back to the project, the aim of our segmentation process is, given a skin lesion image, to return a binary image where pixels belonging to the skin lesion are set to 1 and others to 0. Something like the figure 3.11 (from [22]).

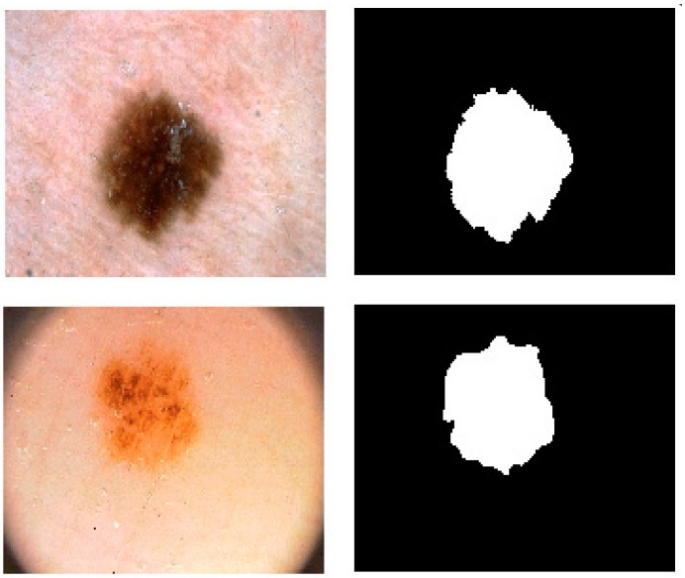


Figure 3.11: Skin lesion segmentation ideal output.

Now, the skin tissue has a property that really jeopardize the task of segmentation: hairs. Hairs presence can't be classified as *noise* or *artifact*, as it is not the result of information loss during acquisition and storage, and it has a counterpart in the physical object being imaged [23]. Hairs introduce new edges and regions in the image that can mislead the segmentation algorithms into wrong conclusions.

In order to visually explain you how misleading hairs are, let's take a look at how the active contours algorithm (ACA) perform under the presence of hairs. The ACA tries to detect the foreground object boundary (read more in 3.3.2) and is so called because it is based on an initial contour that is deformed until it hopefully land into the foreground-background edge, thus being perfect for analyzing the influence of hairs (see figure 3.12).

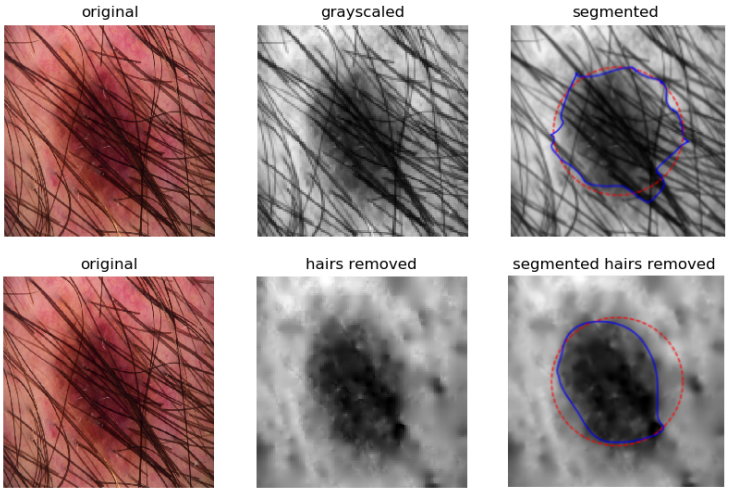


Figure 3.12: original image segmentation vs. hairs-removed image segmentation. In red the initial contour, in blue the final one.

As shown above in figure 3.12, from the red circumference given as the initial contour to be deformed, the blue curve is obtained as the final one, illustrating the output of the ACA, the predicted boundary between the foreground and background. As hairs introduce new edges, (hairs-background mostly) it is easy to see how, without the hairs removal step, the contour gets stuck in those edges, thus producing a wrong output and misidentifying the skin lesion boundary.

Once stated the need of a hair removal process (HR) in order to prevent the hairs presence from influencing the rest of the processes, let's continue in the section below, where I will explain the algorithm that achieves such a smooth removal.

3.3.1 HR

The hair removal process (HR) provides the following conversion:

- Input: an RGB (6000x6000) image
- Output: a grayscaled hairs-removed (128x128) image

The crucial point of this algorithm is to compute the bottom hat operation of the image, which is equal to the image itself minus its *closing*. Let's examine what that means.

Following the article [24], given a grayscaled image, the algorithm steps are described below

- YIQ color space conversion:

  YIQ, which is the acronym for luminance (Y), hue (I), and saturation (Q), is a different standard color space. The first component, luminance, represents gray scale information, while the last two components make up chrominance (color information)) and it has been experimentally proven [24] that hairs are much more identifiable in the Y channel instead of any of the RGB space color (the following images and captions have been copied from [24]):

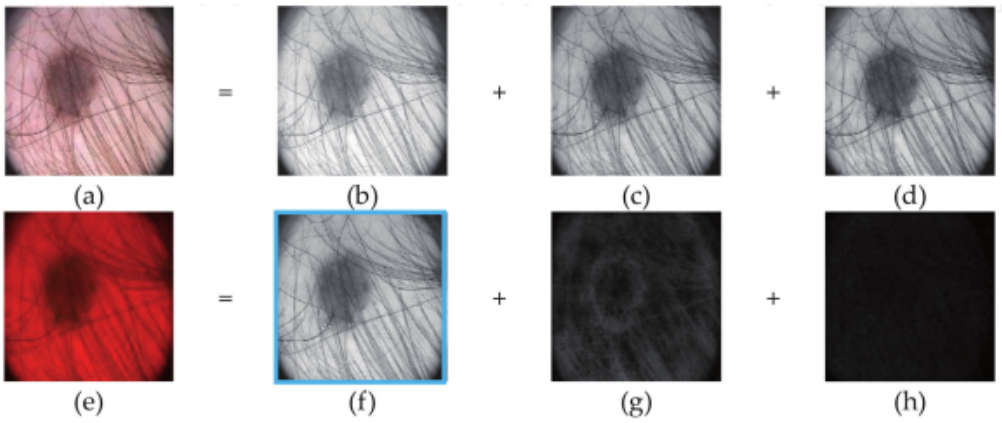


Figure 3.13: A digital dermoscopic image presented in RGB (a-d) and YIQ (e-h) color spaces.

  It is easy to see how the Y channel gives to hairs much stronger relevance than I and Q do, while this difference doesn't happen for RGB channels. This shows how the luminance is more sensitive to hairs.

- Image bottom-hat computation:

  The bottom-hat of an image is equal to the image itself minus its closing. The closing (described in section 2.1.3) is a morphological operation that computes the erosion of the dilatation of white components in an image. Thus, considering the structure we want to isolate, hairs, is typically dark and thin, an image closing should hopefully discriminate those structures, while leaving anything else almost untouched. When the closing is subtracted to the image itself, hairs are completely isolated:

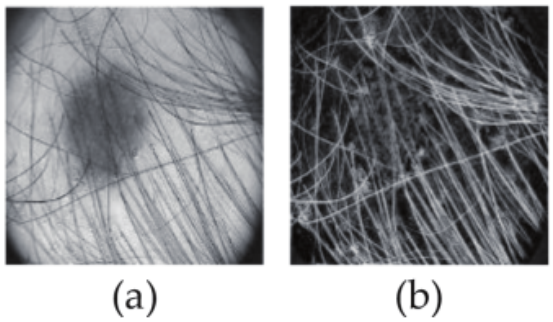


Figure 3.14: The ideal hairs detection result. (a) Y-channel image. (b) Result of bottom-hat operation.

- Hairs mask: The result of bottom-hat is now binarized with Otsu's method, in order to create a binary mask:

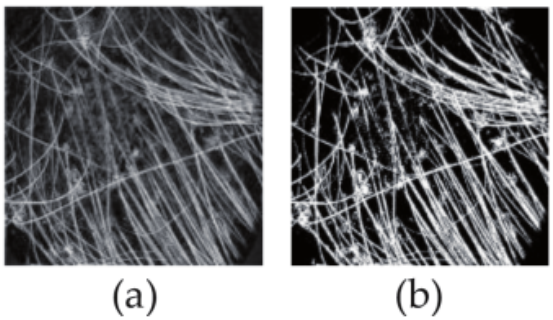


Figure 3.15: Hair mask. (a) Result of bottom-hat operation. (b) Result of Otsu's method.

- Image inpainting:

  This is the process of restoration of a defective image, being in this case the defective regions' pixels the ones where the hairs mask is equal to one. Each defective pixel value is reconstructed with bi-harmonic equations, taking neighboring values and making the transitions as smooth as possible.

Implementation

For implementing the hairs removal algorithm, two attempts have been made to make use of already defined tools from GitHub platform, [25, 26]. However, for my particular case, both of them didn't manage to work out proper hairs mask, so I decided to implement myself a hairs removal algorithm, taking advantage of *skimage* package tools, and following the previous steps I described.

There are actually a few considerations to be done:

- the grayscaled image inpainted is not the result of a common RGB to grayscale conversion. Instead, I took the B channel, because it has been experimentally proved to bring a better segmentation performance than the grayscaled thanks to its contrast improvement [27];
- the hairs mask is black-dilated before the inpainting, so that it can cover properly the hairs regions.

In the figure 3.16, the performance of my HR algorithm on manually selected images is shown, as well as the inner steps previously described, so as to make the whole process more understandable.

And now the question is: is this hairs removal going to benefit the system? Is the segmentation, as well as the other steps, going to be less influenced by the presence of those hairs? Well, if I said CNN classification is going to be improved it would be only a speculation, because I really don't know, but up to the segmentation, we can compare graphically what the performance without and with the HR is (figure 3.17), using the final technique chosen for segmentation, the Chan-Vese algorithm (described in section 3.3.3).

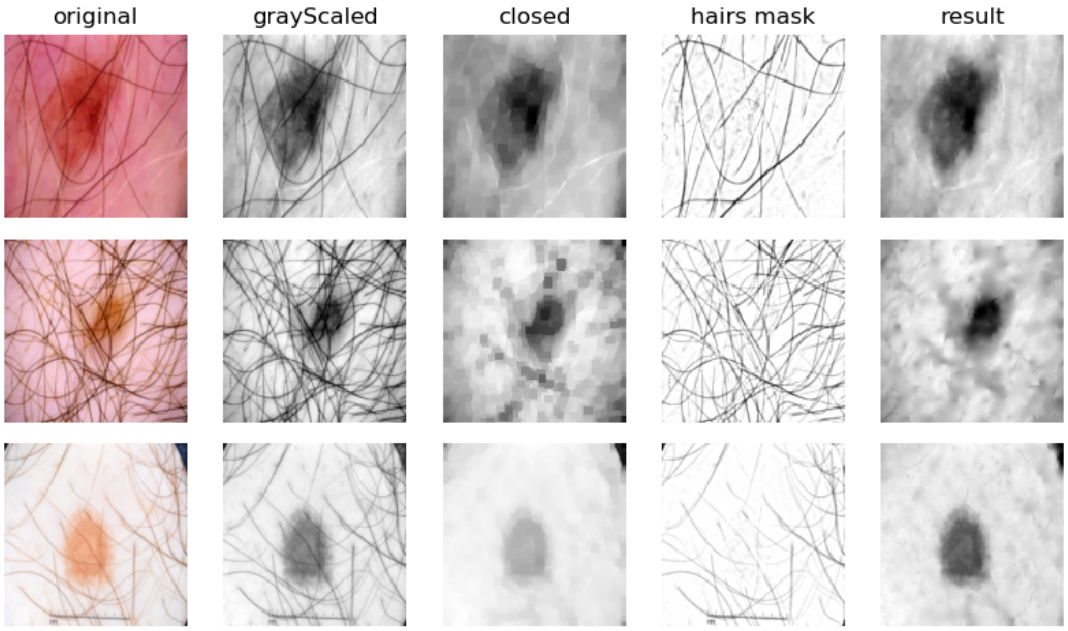


Figure 3.16: Steps of the HR processing on three different images.

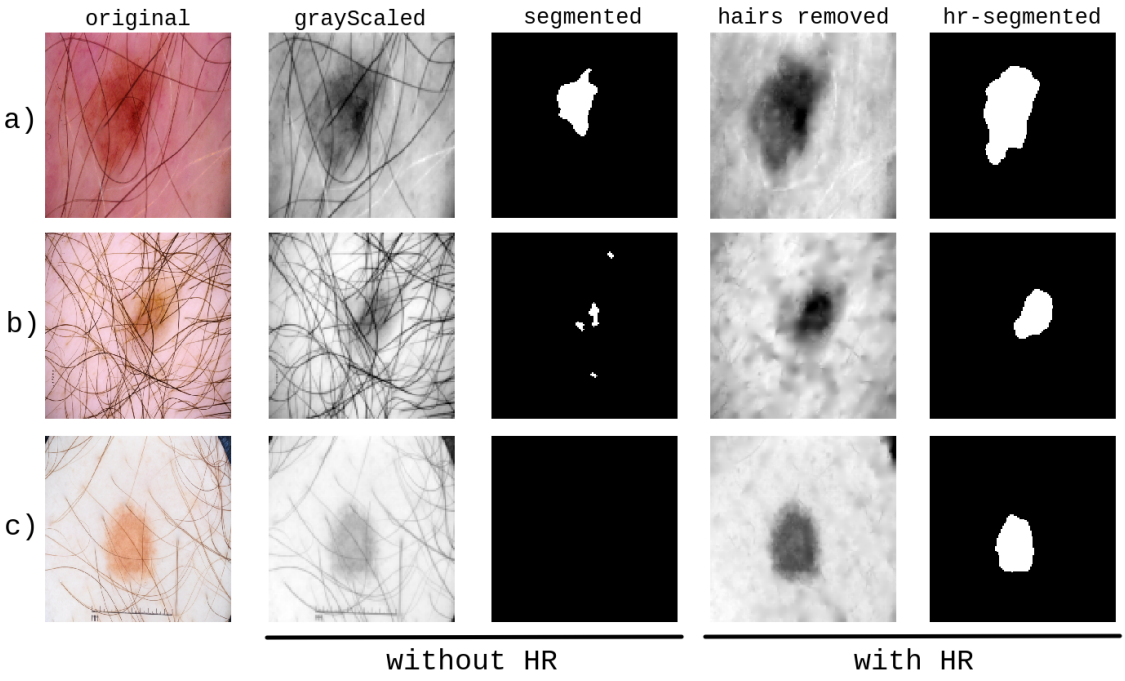


Figure 3.17: Hairs-removed images segmentation vs. original images segmentation. a) Eventually, the segmentation output was simply not complete without the hairs removal step. b) Sometimes, the hairs definitely misled the algorithm. c) Others, it was not even detected any spot at all, while with the improvement of hairs removal it is.

Figure 3.17 illustrates how the hairs removal significantly improves the segmentation process.

3.3.2 Active Contours

The segmentation method used in this project will be the Chan-Vese Algorithm (CVA) [28]. It was proposed in 1999 as "An Active Contours Model Without Edges", *i.e.* as an improvement of an already existent algorithm called *active contours*, so we must first talk about it.

The active contours model, also named "snake", was introduced in 1988 [29] and, quoting its inventors:

"A snake is an energy-minimizing spline guided by external constraint forces and influenced by image forces that pull it toward features such as lines and edges. Snakes are active contour models: they lock onto nearby edges, localizing them accurately. Scale-space continuation can be used to enlarge the capture region surrounding a feature. Snakes provide a unified account of a number of visual problems, including detection of edges, lines, and subjective contours; motion tracking; and stereo matching."

The algorithm consists in the minimization of the energy of an initial curve, also called snake. Considering an image I, and defining the snake as a parametric bi-dimensional curve $C(s)$, like:

$$C(s) = \begin{bmatrix} x(s) \\ y(s) \end{bmatrix}$$

for $s \in [0, 1]$, the energy of the snake $E(S)$ is a function that somehow gives mathematically what our intuition tells us about how good the snake is in terms of segmentation capability:

$$E(C) = E_{int}(C) + E_{ext}(C)$$

Where:

- $E_{int}(C)$ is the internal energy, depending only on C, and gives a score about how elastic and smoothed the snake is. This information is encapsulated in the first and second derivative of C, respectively:

  $$E_{int}(C) = \alpha \int_0^1 |C'(s)|^2 ds + \beta \int_0^1 |C''(s)|^2 ds$$

  being $\alpha, \beta \in \mathbb{R}$ the weights to define the relative influence of each term to $E_{int}(C)$.

- $E_{ext}(C)$ is the external energy, and gives a score on how much C and the image edges match; the image edges information is encapsulated in the image gradient $\nabla I(C(s))$:

  $$E_{ext}(C) = -\gamma \int_0^1 |\nabla I(C(s))|^2 ds$$

  being $\gamma \in \mathbb{R}$ the relative influence of this term against the previous ones.

Therefore:

- The more C shrinks, the lower C' will be, decreasing $E(C)$
- The smoother C is, the lower C'' will be, decreasing $E(C)$
- The more C land on edges, the higher $\nabla I(C)$ will be, decreasing $E(C)$

And vice versa. The solution to the E minimization requires variational calculus; however, in digital images the problem is solved by creating an artificial time t and making the curve $C(s,t)$ change over time until a convergence condition on $E(C(s,t))$ is fulfilled. Of course, integrals are valid for a continuous perspective of the image, and useful for explaining how the algorithm works, but they actually need to be changed to finite series in digital images.

The ACM has been widely used in many applications. While the threshold-based segmentation does not have any knowledge about the contours it is defining, the ACM has the flexibility to adapt to the images edges while maintaining a believable contours of objects, like their real and continuous physic counterpart (macro-perspective-wise, no quantum information segmentation - so far).

The active contours model is again implemented in the *skimage* python package, having the three parameters above described (α, β and w_{edge} respectively), as well as the time step parameter and the convergence condition.

You can see in figure 3.18 how well the active contours algorithm can fit a non geometrical shaped object:

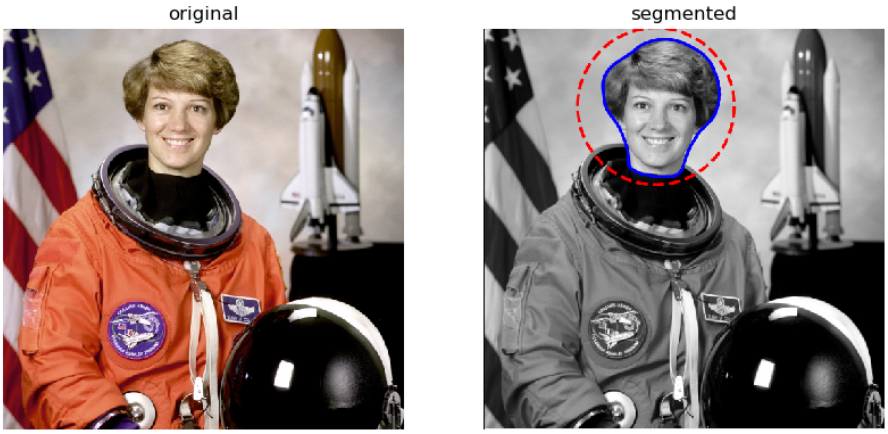


Figure 3.18: ACM segmentation of the astronaut face; the red and blue curves are the initial an final snakes, respectively.

So, if ACM offers so many improvements compared to threshold-based segmentation, why is it not used in this project? Let's take a look below at what the negative aspects of this algorithm are.

Initial conditions instability: the algorithm performance is strongly dependent on the initial snake (look how the red curve in figure 3.18 is so *close to* and *rounding* the object to segment)

The natural movement of the snake is to shrink, thus if the initial contour is not set to initially contain the object, the snake will never fit it because it won't bump

into its edges while shrinking as illustrated in figure 3.19. You can think of making $\alpha < 0$ to make the snake expand, but you will find the same problem, although now with the not contained objects detection.

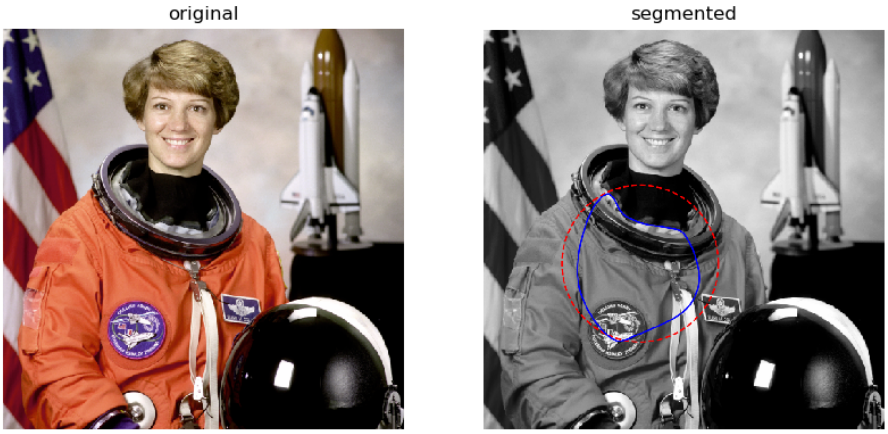


Figure 3.19: Failure of the ACM segmentation when the initial snake don't contain the object to segment (the astronaut face).

Topology changes: the ACM algorithm itself can not deal with more than one snake connected component, as you can see in figure 3.20 (from [30]). Because of its nature, it cannot handle topology features, and if the object has any inner hole, both the external and internal boundaries will never be detected together.

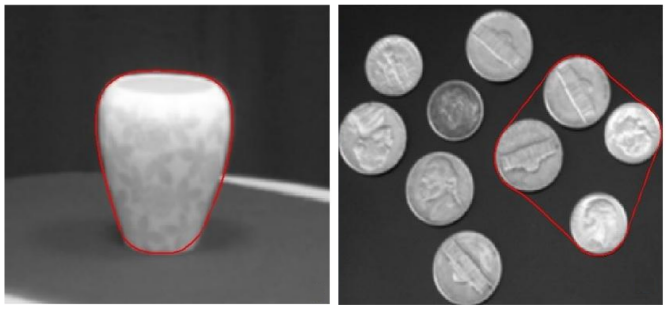


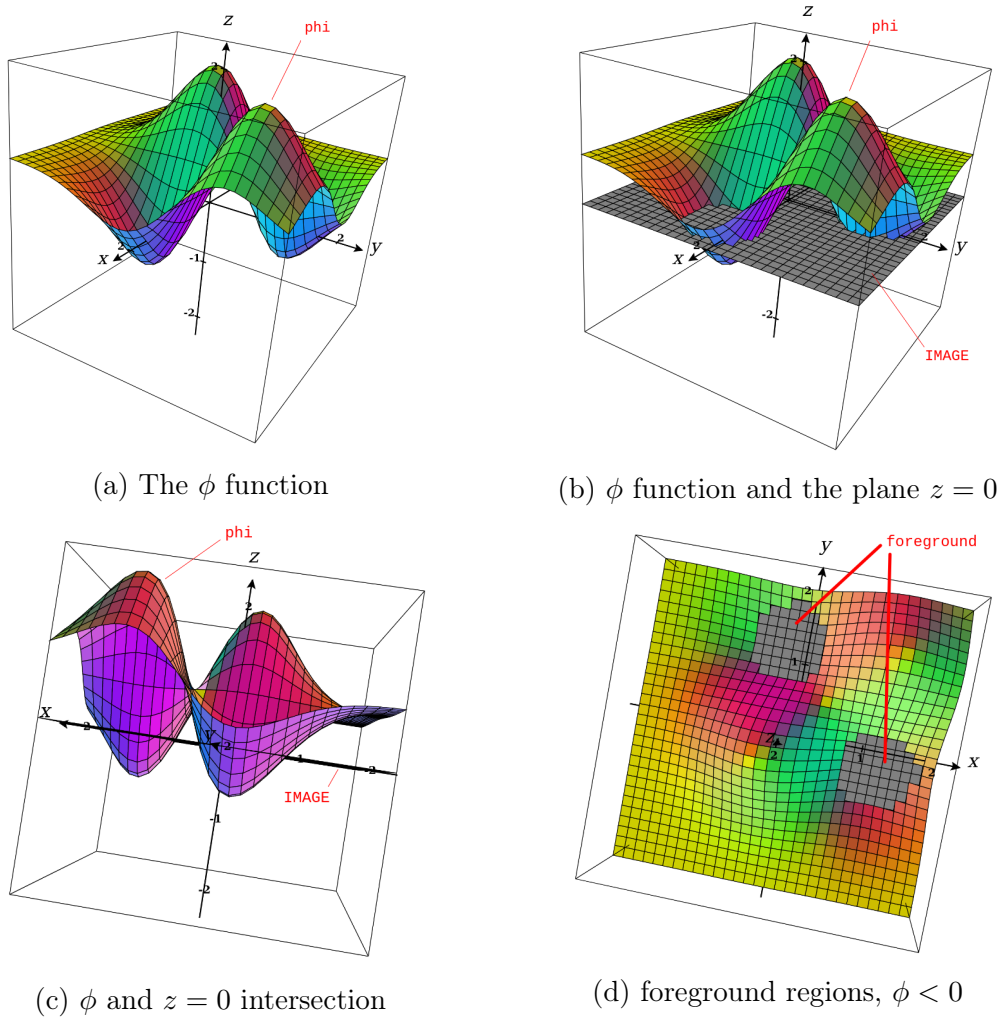
Figure 3.20: Failure of the ACM segmentation in topology changes. The red curves are the final snakes.

You could think that some workaround might exist to set a general initial snake suitable for this project, but skin spots are very different in size, shape, and position in the image. Besides that, what if the mole has more than one connected component?

The ACM can accurately detect contours, but it is only really powerful and useful in a semi-automatic segmentation context or in a problem where objects don't vary enough in shape and position and a reasonable initial conditions cane be set. As long as a human manually draws as many initial contours as needed, containing the objects of interest, this algorithm will fit quite well to the edges, but it is likely to fail otherwise.

3.3.3 The Chan-Vese Algorithm

The algorithm proposed by Chan-Vese can be thought of as an "active surface model", as, instead of a curve, the idea is now to create a function $\phi(x, y)$, being (x, y) the image coordinates, and to evolve it over time [31]. At each instant, the intersection between the surface $\phi(x, y)$ and the plane $z = 0$ (which is the curve $\phi(x, y) = 0$) will implicitly define the object contours, thus also distinguishing both background and foreground regions (see figure 3.21).



(a) The ϕ function

(b) ϕ function and the plane $z = 0$

(c) ϕ and $z = 0$ intersection

(d) foreground regions, $\phi < 0$

Figure 3.21: The Chan-Vese ϕ function visualized.

But how does the surface evolve now? The adopted approach is, like before, an energy function E minimization. This time,

$$\begin{aligned} E =& \lambda_0 \int_{inside} (I(x,y) - \mu_{inside}) dxdy \\ &+ \lambda_1 \int_{outside} (I(x,y) - \mu_{outside}) dxdy \\ &+ \mu \int_{\Omega} |\nabla H(\phi(x,y))| dxdy \\ &+ \nu \int_{\Omega} H(\phi(x,y)) dxdy \end{aligned}$$

Where Ω is the boundary of the curve $\phi(x,y) = 0$; $\lambda_0, \lambda_1, \mu, \nu \in \mathbb{R}$ make the weighted influence of each term; μ_{inside} and $\mu_{outside}$ are the mean of foreground and background pixels respectively; and $H(z)$ is the Heaviside function:

$$H(z) = \begin{cases} 1 & \text{if z} \geq 0 \\ 0 & \text{if z} < 0 \end{cases}$$

The first two terms in E are a penalty for the variance of pixel values belonging to the same region, and so a reward for the uniformity of the foreground and background, respectively. The second and third terms instead, punish the algorithm for the length of the curve and the foreground total area [32].

The Chan-Vese algorithm is similar to the ACM, as the curve $\phi(x,y)$ will spontaneously tend to shrink, while trying to separate regions with uniform pixels values, but it is more powerful because there is nothing preventing it from segmenting several-connected-components regions, while detecting also their holes, as you can see in figure 3.22.

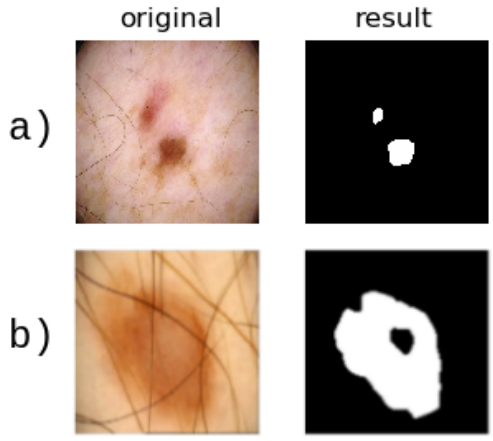


Figure 3.22: Success of Chan-Vese algorithm when segmenting: a) multiple connected components objects, b) objects with holes.

Implementation The Chan-Vese algorithm is also implemented in *skimage*. The function receives only three of the four weights described before, as, in order to simplify the optimization, the ν factor for the total area penalty is not implemented [33].

As in the HR algorithm, there are some steps I added to the segmentation process, to fit it to my specific problem:

- because the skin lesion can be extensive enough to make the algorithm think it is the background instead of the foreground, a circle mask is applied to the image, in order to set to black the furthest from center pixels, and thus make the algorithm always identify the mole as *background* (black).
- a morphological image opening of the binary inversion (see section 2.1.3) (*i.e.*, a black-opening) needs to be done in order to eliminate small false positive components.

Besides figure 3.17, where the Chan-Vese segmentation performance on three hairs-removed images was shown, below you can take a look at an example of each specific step of the algorithm:

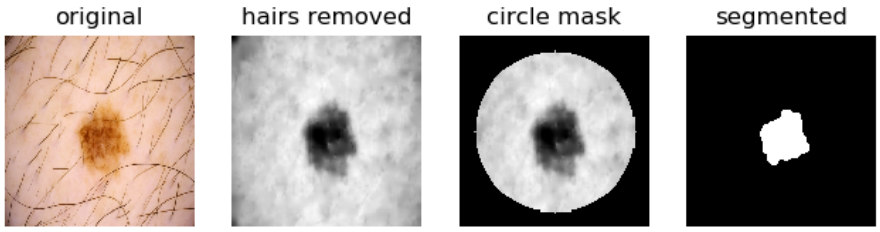


Figure 3.23: Chan-Vese segmentation steps.

Once each image has been segmented and thus the most relevant information can be identified, some interesting geometrical and optical features can be extracted, taking advantage of the segmentation mask. That extraction will be explained in the section below.

3.4 SFE

The Statistical Features Extraction (SFE) process consists in the following conversion:

- Input: original and segmented images.
- Output: a 1D-features vector for each image.

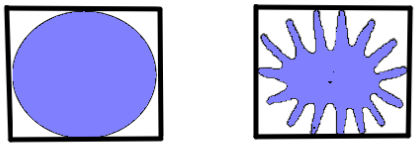
SFE is a simple geometrical-features extraction process that follows the segmentation one. Its goal is to compute, for each image, some shape analysis of the segmentation region and some color statistical analysis of the original image, masked by the segmentation.

About the shape analysis, the idea is to extract, for each image, and for each connected component:

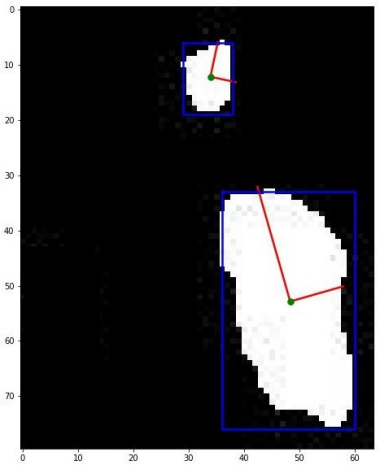
- one centroid,
- the eccentricity of the region,

- the major and minor axes of the ellipse that has the same normalized second central moment, and
- a quantification of the convexity, the areas ratio between the minimum rectangle that inscribes the region and the region;

You can graphically view all this information in figure 3.24.



(a) The area ratio visualized



(b) Major and minor axis of two connected components (red) and the circumscribed rectangle (blue)

Figure 3.24: Shape features visualization.

Besides that information, the red, blue and green channels means and deviation are going to be extracted. Formally, let I be the original image, I^R, I^G, I^B its R, G, and B channels, respectively, S be the segmented region, having k connected components $c_0, c_1, ..., c_k$. Then, the SFE computes:

- *areas ratio*: $\sum \frac{area(c_i)}{area(b_i)}$, where b_i is the rectangle that inscribes c_i.
- *major axis length mean*: $\frac{1}{\sum maj(c_i)} \sum maj(c_i)^2$, where maj is the *major axis length* of c_i.
- *minor axis length mean*: $\frac{1}{\sum min(c_i)} \sum min(c_i)^2$, where min is the *minor axis length* of c_i.
- *eccentricity mean*: $\frac{1}{k} \sum ecc(c_i)$, where $ecc(c_i)$ is the *eccentricity* of c_i.
- *eccentricity standard deviation*: $\frac{1}{k} \sum \sigma(ecc(c_i))$, where $ecc(c_i)$ is the *eccentricity* of c_i.
- *red channel mean*: $\frac{1}{|S|} \sum I^R(S)$.
- *blue channel mean*: $\frac{1}{|S|} \sum I^B(S)$.
- *green channel mean*: $\frac{1}{|S|} \sum I^G(S)$.
- *red channel standard deviation*: $\sigma(I^R(S))$.

- *blue channel standard deviation*: $\sigma(I^G(S))$.
- *green channel standard deviation*: $\sigma(I^B(S))$.

One important consideration should be done to explain why I decided to extract the weighted mean for certain features and not for others. The following paragraphs will clarify the reason for this choice. S can have many little connected components, that can deviate the *major* and *minor axis length mean* without providing significant information, which is why I decided to compute a weighted mean $\mu*$, where the weight of each term is proportional to itself, *i.e.* $w_i = \frac{1}{\sum x_i} x_i$, and so:

$$\mu * (X) = \sum w_i x_i = \sum \frac{1}{\sum x_i} x_i^2 = \frac{1}{\sum x_i} \sum x_i^2$$

the weighted sum achieves the goal of correcting the influence of each term, by weighting it with a value equal to itself, which hopefully is the relevance of the counterpart physic object.

You can think the same approach may apply for the *eccentricity mean* and *standard deviation*, but it does not, since eccentricity is a property relative only to c_i shape, and it is not *extensive* thermodynamic-wise speaking because it doesn't depend on the size the object it is describing.

Implementation Again, the *scikit-image* provide a useful region shape analysis function, *regionprops*, that automatically extracts all the shape features described above.

3.5 ECE

The Euler Characteristic Extraction (ECE) process consists in the following conversion:

- Input: hairs-removed images.
- Output: a 1D-features vector for each image.

Topology is the branch of mathematics that study the properties of an object that are preserved under continuous deformations, such as stretching, twisting, crumpling and bending, but not tearing or gluing [34].

From a topological perspective, a cup and a doughnut are the same, as one can be deformed into another without any tearing or gluing (figure 3.25).

Figure 3.25: Deforming a mug into a doughnut.

But if they are so, then what things are they not equal to? What makes those objects any different from a sphere, for instance? The answer is the hole. A compact sphere is a unique connected component without any holes or cavities, while a doughnut (a torus) has also one connected component, but two tunnels and one cavity. All the information about the connected components, tunnels and cavities, can be stored in a single (sort of magic) number: the Euler's characteristic.

3.5.1 Understanding of Euler's Characteristic

The Euler characteristic χ was classically defined for polyhedral surfaces as:

$$\chi = V - E + F$$

where V, E, and F are the numbers of vertices, edges, and faces respectively. That formula is valid and useful as long as the problem stays within the third dimension. The general formula for the χ of a given complex is given by:

$$\chi = \sum_{i=0}^{n} k_i$$

where n is the dimension of the considered complex and k_i is the number of i-dimensional cells of the complex. The definition of "complex" is quite hard to understand without enough mathematical background, but you can think of it as an nD generalization of polyhedra. Likewise, hard to graphic, but you can think of i-dimensional cells as a generalization of points, curves, surfaces, volumes, hypervolumes, (0, 1, 2, 3, 4-dimensional cells respectively) and so on.

I have been studying for over a year how the topological features of an image can be extracted and used, and ended developing a toolkit for a simple topological analysis of nD images in python (a.k.a. TANI). The toolkit has two main functionalities: extracting a topological histogram and analyzing textures in different sort of ways, both applied on grayscaled images. The project is available in GitHub [35], and I am introducing it because I will reuse a simplified version of the first functionality, as I am going to extract a vector of χ values from each image.

The proposed algorithm of Euler's characteristic extraction (ECE) will firstly digitize the image into five different gray values, and then extract the χ of each one of them, acquiring for each image a vector of five integers representing the $\chi(g)$, for $g \in G$ being $G = 0, 0.25, 0.5, 0.75$, *i.e.* the set of different gray values of the digitized image.

3.5.2 n-Cells Identification Model

Thanks to the notation used in TANI [36], different-dimensional elements can be easily handled as same-shaped arrays. Let's see how.

Let us consider an n-dimensional space, where every point can be identified by its coordinates:

$$(u_0, u_1, \ldots, u_n)$$
$$u_i \in \mathbb{N}$$

Let J be an extension of V such that "point density is doubled" and therefore coordinates can be multiple of $\frac{1}{2}$:

$$(\frac{1}{2}u'_0, \frac{1}{2}u'_1, \ldots, \frac{1}{2}u'_n)$$
$$u'_i \in \mathbb{N}$$

Let $P \in J$ be the point whose coordinates are:

$$(x_0 + \lambda_0\frac{1}{2}, x_1 + \lambda_1\frac{1}{2}, \ldots, x_n + \lambda_n\frac{1}{2})$$

with

$$x_i \in \mathbb{N}$$
$$\lambda_i \in 0, 1$$

Such coordinates will no longer identify P; they will instead refer to the r-dimensional entity ("*cell*") E with:

$$r = n - \sum_{i=0}^{n} \lambda_i$$

P happens to be "the mass center" of E. Note that:

- when $\lambda_i = 0 \, \forall i$ then $dim(E) = n$, so E is not a point.
- when $\lambda_i = 1 \, \forall i$ then $dim(E) = 0$, so E is a common 0-dimensional point.

Thanks to this notation, a set of equal-length arrays can describe a set of cells of different dimensions that form a unique object. For instance, a volume can be described as a set of arrays having all its 3D cubes, 2D surfaces, 1D edges, and 0D vertices. I hope you are seeing by now how subtly powerful this can be when it comes to compute the Euler number: for this purpose, the χ of an object can be computed using simply the generic formula.

3.6 AI techniques applications

Along this project several image processing and numerical features extraction methods have been described, each one of them producing its outputs from some inputs. Now it is the time to study the relationship between all the input, inner, and output data we have worked on, and it is in this context that AI techniques helps.

In order to carry out this analysis, the data can be classified into two types: categorical or numerical, and images. For both categorical and numerical data, an MLP system can be applied, while a CNN is proper for images, as explained in 2.2.4. Recall what kind of data each process produced out of a single image:

- HR: hairs-removed grayscaled image.
- SEG: segmented binary image
- SFE: 1D-vector of eleven float elements
- ECE: 1D-vector of five integer elements

And besides those, from the initial *.csv* file (see 3.1), we have the *sex*, *anatomical site*, and *age*, which we will call *primal variables*; so there are a total of three categorical and numerical dataframes (primal, SFE output and ECE output), which means that six different MLP can be generated, one upon each one of their possible combinations.

Likewise, there are four possible types of images to be taken as inputs (RGB, grayscale, HR, and segmented images), so that four different CNNs can be built, but now the computation effort is so expensive that I couldn't manage to have enough time to test all of them, and I only opted to tested the input that should probably achieve the best results: RGB images set, which is supposed to contain all the others' information. In what follows, we will call all these possible MLPs and CNN, based on a specific kind of data, *sub-models*.

But, why sub-models? is it sensible to analyze *every* combination of such dataframes? Isn't it redundant to build a predictor system out of a subset of data? Why don't we just consider simply all variables? Well, there are two reasons why it may be sensible to do so:

- to satisfy the science hunger of knowledge: whichever the better model could be, study other (worse) ones can lead us to important thoughts and conclusions about the data and the usefulness of the process that generated it, and
- because it is strictly not guaranteed that the more input variables you have the better your MLP is going to predict. When it comes to the input variables, especially in this project where, even being of different natures (shape, color, topology), they come all from the same images, redundancy can happen. And if a physical property of an image is reflected in more than one variable, then the influence of that particular property over the system would be doubled, thus introducing a bias in the prediction, which can worsen it compared to other MLP models based on less inputs. *Redundancy* is not equal to *bias introduction*, but it can cause it.

Below, I will explain the principle that I have followed to design the networks architectures of this project.

3.6.1 Between Under and Overfitting

Quoting [37],

"The fundamental issue in machine learning is the tension between optimization and generalization. Optimization refers to the process of adjusting a model to get the best possible performance on the training data, whereas generalization refers to how well the trained model performs on data it has never seen before. The goal of the game is to get good generalization, of course, but you don't control generalization; you can only adjust the model based on its training data.

At the beginning of training, optimization and generalization are correlated: the lower the loss on training data, the lower the loss on test data. While this is happening, your model is said to be underfit: there is still progress to be made; the network hasn't yet modeled all relevant patterns in the training data. But after a certain number of iterations on the training data, generalization stops improving, and validation metrics stall and then begin to degrade: the model is starting to overfit. That is, it's beginning to learn patterns that are specific to the training data but that are misleading or irrelevant when it comes to new data."

The overfitting is the phenomenon that occurs when a network stops learning on inputs, to memorize them (see figure 3.26). The ultimate goal of intelligent classifier systems is to learn on a training set, while being general enough to predict with certain reliability the class for new cases. When a network starts learning, the updating on its weights, accomplish to understand what the complex relation between class an variables are, getting step by step closer to the general problem. When enough time has passed though, the network eventually starts memorizing inputs, and its predictive capability is optimized for the training set only, thus taking steps back in generalization. The instant in which the overfitting is considered to begin is the instant when the systems achieves the minimum validation loss (see 3.6.7).

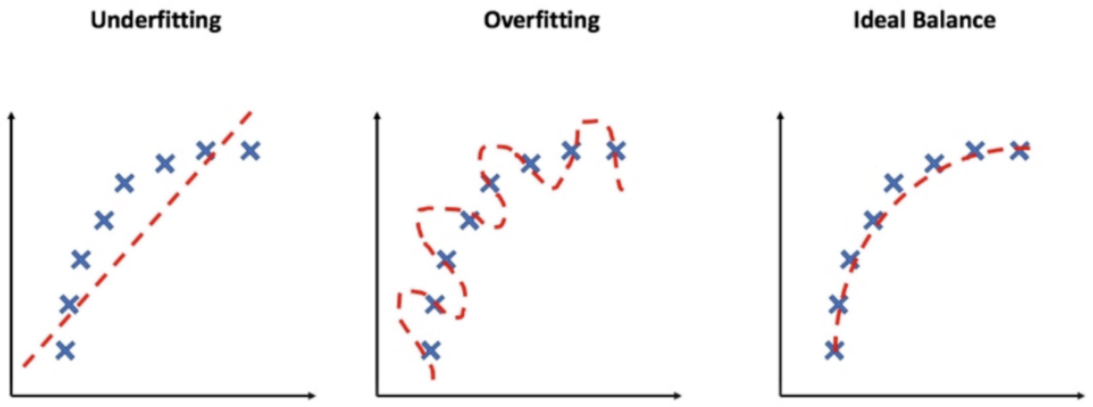


Figure 3.26: Under and Overfitting

The process of fighting against overfitting is called regularization. Some regularization techniques are:

- Reducing the network size: the more layers and inputs in them has a network, the better its memorization capability will be. And, forgive me if I quote again [37], but it is really worth it:

" [...]. For instance, a model with 500,000 binary parameters could easily be made to learn the class of every digit in the MNIST training set: we'd need only 10 binary parameters for each of the 50,000 digits. But such a model would be useless for classifying new digit samples. "

- Weight regularization: inspired by the idea of having a model where the distribution of parameter values has less entropy, this factor adds a cost function associated with having large weights. The cost function can be:
  - Proportional to the weights absolute values (L1 regularization)
  - Proportional to the square of weights values (L2 regularization)
- Dropout: applied to a layer, randomly silence some units, thus taking some information away from the network and preventing it from memorization.

So, a theoretical balanced point between under and overfitting is the aim of a network training. Below I will describe the architectures I designed to try to reach that ideal equilibrium point.

3.6.2 Systems Design

Each submodel share with others of the same kind (MLP or CNN) its input and output layers. Besides that, five degrees of freedom have been considered for the architecture:

- Inner layers units
- Dropout layers
- L1 and L2 regularization layers
- Batch size
- Epochs

3.6.3 MLPs

For MLPs, simple dense architectures have been used, with a Dropout and regularization layers to fight against the overfitting, as explained above. The basic input and output layers structure will remain constant in each model and will be the responsible for the normalization and hot-encoding of numerical and categorical data, respectively:

- numerical data: numerical variables provided as inputs are often from different magnitude orders which causes the network to be more influenced by some of them than others. The generally used technique is to normalize the data, so as to overcome the differences.
- categorical data: a categorical variable is the one whose values are strings, but the network works with numbers, so, how are those variables accepted as inputs? The technique used to face this issue with may depend on the type of categorical variables:

- ordinal variable: values can be put in a logical order (for example the level of education completed: High school, college, Bachelor's degree, etc). Values can be encoded as integers, observing the logical order.
- nominal variable: they are labels, and are not spontaneously sorted (for example colors). Those variables can't be encoded as integers, as, if so, the model will assume there is a natural order between them. Instead, they are encoded using the *one-hot-encoding* technique, that consists in, being n the number of categories, translating each value into a vector $\mathbf{v}$ with $n-1$ elements, where each v_i represents a category and can be equal to 1 (if the value is of that category) or 0 (if not); see figure 3.27, from [38].

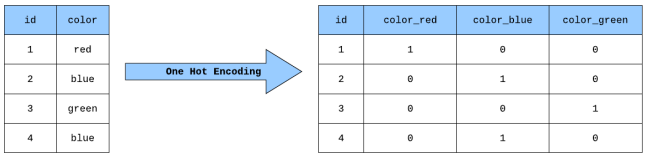


Figure 3.27: One hot encoding technique.

In order to execute the preprocessing that each variable should undergo for the inputs set to be balanced and equitable, *Keras*, the framework used in this project to design AI systems, implements special layers that perform the normalization and hot encoding, accepting only one input. So, the basic structure of input-output of the MLP models will be:

- Input architecture:
  - for each specific accepted input of the system, a one-unit input layer will be built. It guarantees the numerical variables normalization and the categorical values one-hot-encoding, as explained above.
  - a concatenation layer: that accepts and unify all normalized and encoded inputs.
- Output architecture: a 1-unit densely connected layer with sigmoid activation.

From now on, all I am going to mention are the inner layers; the basic input-output structure described above is always implicit. The general models I have experimented with, following the heuristic described in 3.6.1, are:

Architectures

- *model0*: proposed to underfit
  - 6-units densely connected layer
  - 2-units densely connected layer
- *model1*: proposed to overfit

  - 32-units densely connected layer
  - 16-units densely connected layer
  - 8-units densely connected layer

- *model2*: overfit correction with regularization factors
  - 16-units densely connected layer
  - 8-units densely connected layer with $l_1 = 0.001$, $l_2 = 0.001$
  - Dropout 0.3

- *model3*: overfit correction with higher regularization factors
  - 16-units densely connected layer
  - 8-units densely connected layer with $l_1 = 0.01$, $l_2 = 0.01$
  - Dropout 0.5

- *model4*: overfit correction with size reduction
  - 12-units densely connected layer with $l_1 = 0.001$, $l_2 = 0.001$
  - 4-units densely connected layer with $l_1 = 0.001$, $l_2 = 0.001$
  - Dropout 0.5

- *model5*: overfit correction with inputs layer dropout
  - 12-units densely connected layer with $l_1 = 0.001$, $l_2 = 0.001$
  - Dropout 0.5
  - 6-units densely connected layer with $l_1 = 0.001$, $l_2 = 0.001$
  - Dropout 0.5

So, for instance, the complete architecture for *model0*, shown in left-to-right configuration, can be seen in the following *url*: `https://raw.githubusercontent.com/agiulianomirabella/melanoma-detector/master/images-utils/MLP_architecture_sample.png`.

Different Activation Functions.

All the units will be by default activated by a *relu* function (see 2.2.2), but the best models are going to be tested with different activation functions too, such as *tanh*, and *sigmoid* function.

Class-balancing.

The architectures that obtained the best performances will also be tested for the second *data strategy* flavor, the 5509 cases with class imbalance correction factors [39].

3.6.4 CNNs

Typically, CNNs have a much more expensive computational cost, thus fewer combinations of architectures and hyper-parameters can be tested in reasonable time taking into account the practical restrictions. To quickly optimize the results, a technique called *transfer learning* has been used instead.

Transfer learning

This technique consists in applying, for a certain problem considered, a system that has already been trained in a different one. Instead of designing the architecture and initialize every weight to a random number, transfer learning relocates the *network knowledge* to be used in another system (figure 3.28, from [40]).

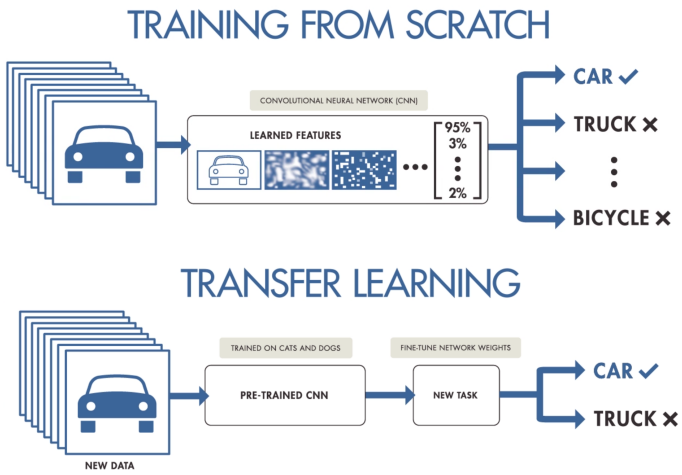


Figure 3.28: Transfer learning vs. learning from scratch.

Keras provides many pre-trained models in its applications module [41].

Architectures

Same as before, only inner layers will appear in the diagram, and in each architecture, one inner layers will always be a model name that accomplish the transfer learning:

- *model0*:
  - DenseNet201 model
  - 16-units densely connected layer with $l_1 = 0.001$, $l_2 = 0.001$
  - Dropout 0.5
- *model1*:
  - MobileNetV2 model
  - 16-units densely connected layer with $l_1 = 0.001$, $l_2 = 0.001$
  - Dropout 0.5
- *model2*:

- EfficientNetB0 model
- 16-units densely connected layer with $l_1 = 0.001$, $l_2 = 0.001$
- Dropout 0.5

Therefore, the *model0* architecture, for example, can be seen here: `https://raw.githubusercontent.com/agiulianomirabella/melanoma-detector/master/images-utils/CNN_architecture_sample.png`.

Data Augmentation

In order to fight the small-sized dataset, a technique called *data augmentation* has been used. It consists in applying randomly certain DIP functions to images, such as rotation, zoom, width and height shift, shear and horizontal flips. Thanks to data augmentation, the images dataset will be augmented up to many more images, obtained by changing randomly the original ones.

3.6.5 Hybrid Model

The proposed hybrid model is an MLP based on a new dataframe, composed by all the numerical and categorical data, but including a new column containing the CNN predictions.

The idea is to join both systems, based on the knowledge obtained from image pixels and features extracted, to forge something new that will hopefully perform more accurate predictions than its parents.

A fake deceptive procedure

A CNN is going to be trained on original images dataset. Later, that CNN will be used to predict the target on the same dataset, generating a new numerical data per image, thus a new column. However, predicting the target on data the system has been already trained upon sounds a little suspicious, doesn't it? Is it really licit?

The answer is yes, *it is licit*, but the real worrying question to rise is: *is it useful?*.

- *It is licit*, because it simply *doesn't break the rules*. The "forbidden" procedure is to give the network some hint about the test data target (please understand *test* as the holdout technique is being used). If the network is given any information about the test dataset, its scores on the test data are not reliable anymore since it has already had knowledge about it. So, the new column obtained by CNN predictions is legitimate as long as the CNN is not trained on the test images.
- But, *is it useful?* how should the dataset be split in order for the system to really learn from it?
  - Suppose we split the dataset in *train* and *test*, the CNN is trained on the *train*, and, once finished, it predicts on the *train* and *test*, generating new columns. Now, doubts arise when the legitimacy of predictions in *train* is questioned, since the system has already visited *train* cases, whose output it is predicting. An already visited case does not imply that the system

shall return its correct output, this would be true only if the accuracy on the training set was 1. But if the MLP is trained on those *fake predictions* (let's call them so), will it learn to generalize the problem enough to be able to use wisely the column of *real predictions* in the *test* dataset?

- There seems to be a workaround to the previous problem. Suppose I split the dataset in *train1*, *train2* and *test*, the CNN is trained on the *train1*, and once it has finished it predicts on the *train2* and *test*. Now the MLP is trained on *train2*, whose CNN predictions are not *fake*, and tested on *test*.

The problem pointed out above is not trivial and requires some timing to be digested. Once stated the system submodels are never going to visit the test data, it is no longer a matter of rightfulness or legitimacy of the experiment, but of data strategy instead.

Unfortunately though, the shortage of the project dataset completely rules the second option out, as the submodels wouldn't have enough cases to learn from. Our unique chance is to train the final MLP system on the CNN predictions as explained above.

Architectures

Architectures similar to MLPs submodels have been used for the final MLP system, with of course a bonus experimental emphasis on those which seemed to get better results. The proposed architectures have been:

- *model0*:
  - 32-units densely connected layer with $l_1 = 0.001$, $l_2 = 0.001$
  - 16-units densely connected layer with $l_1 = 0.001$, $l_2 = 0.001$
  - Dropout 0.5
- *model1*:
  - 32-units densely connected layer with $l_1 = 0.001$, $l_2 = 0.001$
  - 8-units densely connected layer with $l_1 = 0.001$, $l_2 = 0.001$
  - Dropout 0.5
  - 2-units densely connected layer with $l_1 = 0.001$, $l_2 = 0.001$
- *model2*: overfit correction with regularization factors
  - 22-units densely connected layer
  - 8-units densely connected layer with $l_1 = 0.001$, $l_2 = 0.001$
  - Dropout 0.5

Which, again, lead to this sample architecture: `https://raw.githubusercontent.com/agiulianomirabella/melanoma-detector/master/images-utils/HYBRID_architecture_sample.png`.

3.6.6 Experimentation

I will explain in this section what method has been chosen to conduct every experiment and how the performance of each experiment has been evaluated.

Holdout validation

This is the method used for the Hybrid Model. The holdout technique consists in simply splitting the dataset into *train* and *test* set. In order to reduce possible bias though:

- the class balance in both *train* and *test* sets is guaranteed, and
- three different holdouts per each experiment are going to be done, and the final metrics score will be the mean of all scores.

Stratified K-fold validation

For tuning the hyper-parameters of submodels, (MLP and CNN), a stratified 5-fold validation experimentation system has been designed to produce a plot of the means of *accuracy*, *AUC*, and *loss* (see 3.6.7 for the explanation), both for the training and validation dataset, for each experiment. Of course, the relevant ones are the validation accuracy, AUC and loss, since the train ones are quite falsely correct.

But what is the k-fold validation? In holdout technique the dataset is split into training and test sets. The training set tunes the network weights, while the test set is for experimenting with different architectures and hyper-parameters such as batch size or regularization factors. The problem when splitting a dataset into training and test sets is that there is no knowledge about how the target and variables are distributed within both of them. Let's say we want the test set to be the $x\%$. In such case, the measured performance of the system can vary depending on what particular $x\%$ of dataset you choose as validation set.

The k-fold validation algorithm offers a statistical solution to this problem, as it splits the training set in turn into k partitions, creates k models, trains them upon k-1 folds and validates it on one fold, each at a time. The result of the experiment is a mean of all evaluation metrics for each model, that has been trained in each possible combination of folds (see an example for $k = 3$ in figure 3.29, from [37]).

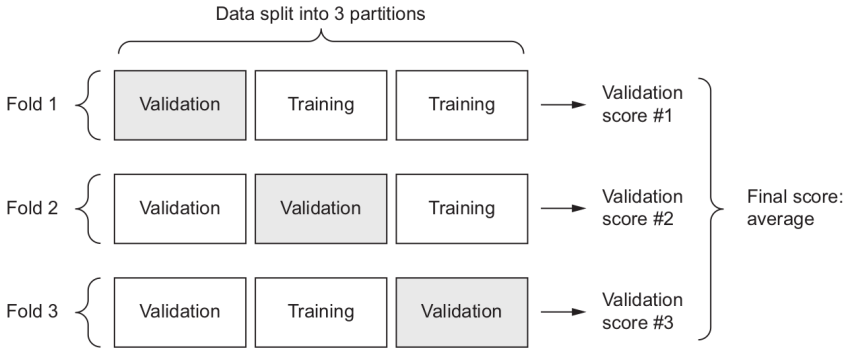


Figure 3.29: K fold validation.

With such algorithm, the system evaluation of the experiment is now much less vulnerable to variables imbalanced distribution than it was before, where only one model was trained. Besides that, the "Stratified" first name is a variation of K-fold that forces the target distribution to maintain quite the same in each fold. Now, it is still possible to get biased performances, but it is much less likely to happen.

Hyper-parameters Tuning

The workflow of the experimentation is based on a *trial and error* philosophy. For this purpose, I have developed a simple experimentation system, that generates, for each experiment, a plot like the one in figure 3.30.

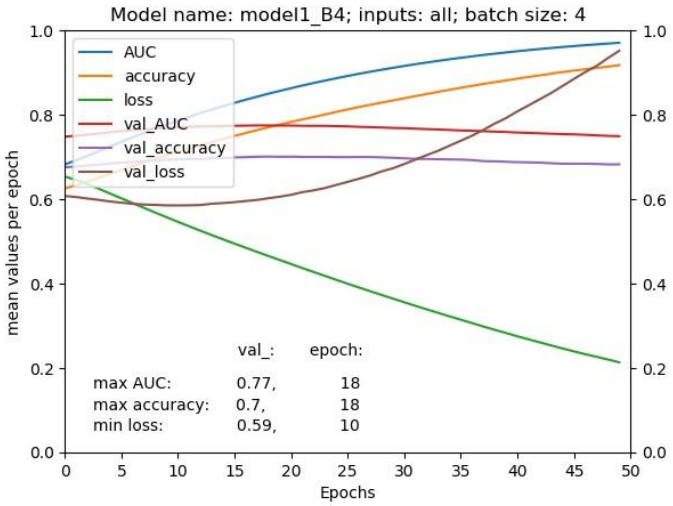


Figure 3.30: Experiment output plot example.

where every relevant info can be visualized: *model architecture*, *inputs* and *batch size* in title, and the maximum validation AUC and accuracy, as well as the minimum validation loss values and situation in temporal (epochs) axis. In the next section I will explain those concepts carefully.

So for each submodel architecture, an experiment is done and, from its output a new architecture or hyper-parameters modification is proposed, leading into a new experiment. For each experiment, a range of batch sizes from $2^1 = 2$ to $2^7 = 128$ is tested and epochs are manually adjusted to make sure the extreme values are reached within the last epoch.

The goal of all this process is to obtain optimal results for each different degree-of-freedom combination. Below, I will explain what the metrics are for quantifying how good an experiment result is.

3.6.7 Evaluation metrics

The evaluation I have conducted on each MLP model has been based on three values: the *loss*, the *area under the ROC curve* (AUC) and the *accuracy*.

- the *loss* is the function the network is trying to minimize in order to learn. In our case, the best loss function to be chosen is the *binary cross entropy*, which measures how much dispersion there is among predictions and actual values differences in each class.

  The train loss function is usually not supposed to stop decreasing, as the system shall learn on training data, and eventually memorize the cases; on the other hand, the validation loss starts decreasing up to the moment when the systems starts overfitting, and thus more predictions on validations cases become wrong, and the loss higher. In fact, the moment when validation loss takes its minimum value is considered to be the start of the overfitting phenomenon.

- the *accuracy* measures how often the algorithm classifies an input correctly. It may seem simple as that, but the problem is that our MLP and CNN systems do not output a single integer number 1 or 0, but instead they provide a floating value between them, which will after be interpreted as a classification, thanks to a threshold. Suppose our predictor produces an output of p, then:

$$prediction(p) = \begin{cases} malignant & \text{if p} > \text{t} \\ benign & \text{if p} \leq \text{t} \end{cases}$$

But, how to set t? As really well explained in [42], as the threshold vary, so do both true and false positives and negatives, and so true positive rate (TPR) and the false positive rate (FPR):

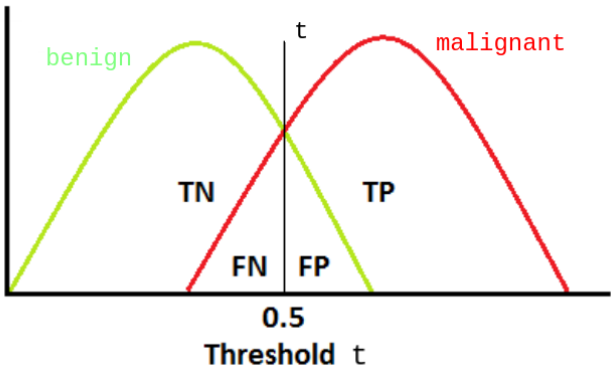


Figure 3.31: TPR and FPR depend on t.

In other words, t variation displace those ratios, making the system more inclined to predict one output rather than the other. So, how can the performance of a predictor system be studied without any influence of t? The answer is to study all possibles values for t, which is why we need the AUC metric.

- AUC: the AUC is the area subtended under the receiver operating characteristic (ROC) curve.

The ROC curve is the graphic obtained by plotting TPR against FPR at various decision-threshold values:

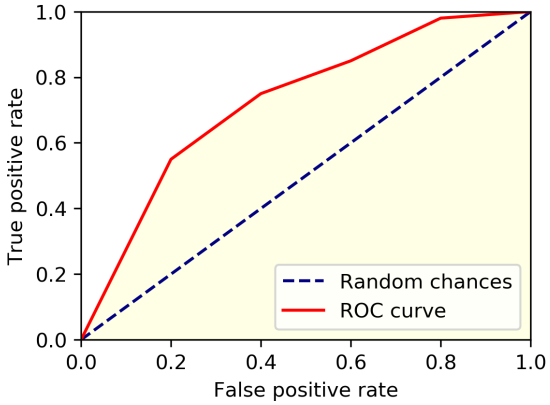


Figure 3.32: The ROC curve (red) and the AUC (light yellow).

The AUC informs about how good the system is predicting at each possible value of t. You can imagine the ROC curve being plotted while t vary, starting from $t = 0$, when all inputs will be classified as 1, (the top right extreme of the ROC curve) up to $t = 1$, when all inputs will be classified as 0 (the bottom left extreme). A random classifier would obtain ~ 0.5 AUC, while an acceptable one should be near the ~ 0.8 value. If the system is performing a < 0.5 AUC then it must be reciprocating the classes, misclassifying the cases.

Once stated and hopefully well described the methods adopted for the project, in the chapter below I will present the results of the application of the described AI techniques and we will discuss their performance as predictor systems, strengths, and vulnerabilities. Then, in chapter 5.2, we will talk about future possible works that may be done having this project as a start.

Before we go to results, there are some considerations to be done in order to explain several technical aspects I learned in this project, which have not been mentioned along it yet.

3.7 More Technical Aspects

The purpose of this section is to describe the technical approach adopted to solve some computation processes along the project. It's been left the last because it is the conceptually less interesting, but it is definitely not negligible, since it has taken the highest percentage of the project time.

3.7.1 Server Support

In order to conduct every DIP process, as well as the CNNs training, the support of an external server was needed. It was provided by the Computer Science and Artificial Intelligence Department of the University of Seville, and its correct and useful working has been achieved thanks to:

- Python version adaptation: Python 3.7 was installed and an anaconda virtual environment was created in order to solve *many dependencies problems*.

- CPU/GPU machine: we had to rely on CPUs only until middle August of this year, but when GPUs were finally available, our time productivity was vividly increased.

- *ssh* and *scp* protocols, in order for my machine to interchange information with the server.

The server was really helpful, providing me with the chance to launch DIP processes during nights, as well as CNNs expensive training that my PC couldn't definitely handle (literally, it fell down several times during some experimenting). However, it is also important to keep in mind how much I had to struggle to take advantage of that help, which is why I am writing this paragraph. It really took me, my tutor and his department colleague, to whom I am very thankful for their time, up to several weeks to solve all the packages and versions discordance.

3.7.2 Python Programming

As I've mentioned before, the project code can be found in GitHub [43]. To me, there are here several coding learned skills to mention:

- scikit-image package: which I actually started becoming familiar with during all this course.

- Keras & tensorflow: probably, both the most powerful and frustrating coding of all the project. It offered such a really intuitive high level AI tools, while also expecting from you a really good control of tensors, arrays, and other Python structures, as well as patience to deal with the documentation of the different versions for GPU correct usage.

- scipy.ndimage module: used for morphological and topological analysis along the project.

Laid all the cards on the table, let's continue with the results obtained by the proposed methods.

Chapter 4

Results & Discussion

In this section I will firstly present the results obtained from every experiment I made with MLP and CNN models, then I will compare them and discuss about the general performance of each one.

4.1 Results

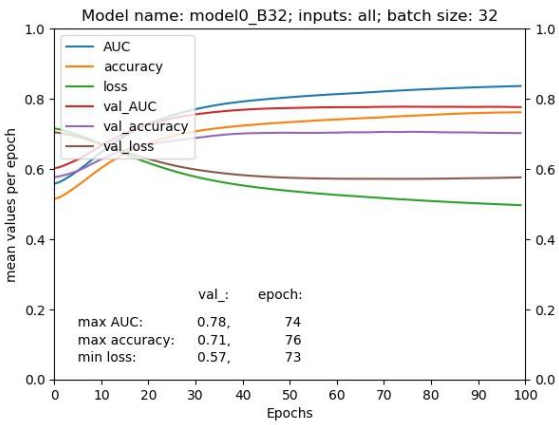
I am going to present, for each model architecture, only the experiments that obtained the best scores in terms of *AUC* and *accuracy*.

4.1.1 MLPs

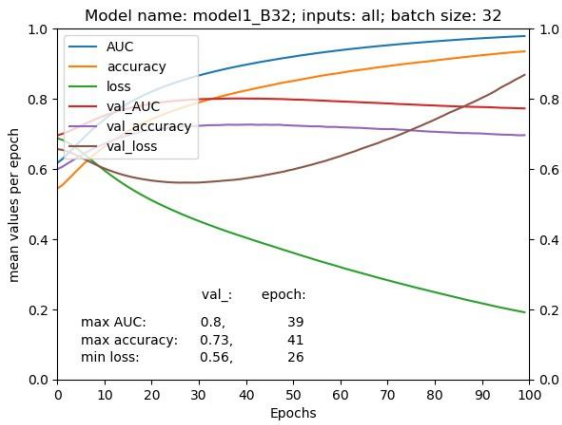
To start with, the results obtained for different MLP systems will be presented.

Complete-inputs-based

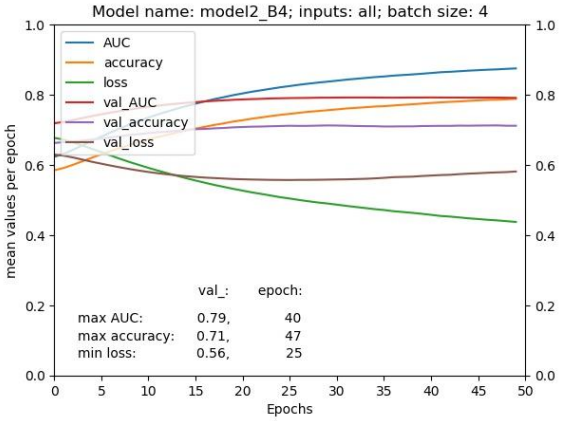
The MLP systems based on the complete inputs sets were those which exploited each possible column of the dataframe obtained by several categorical data from the initial *.csv* file, topological and statistical features extractions process. The performances obtained for MLPs systems are:



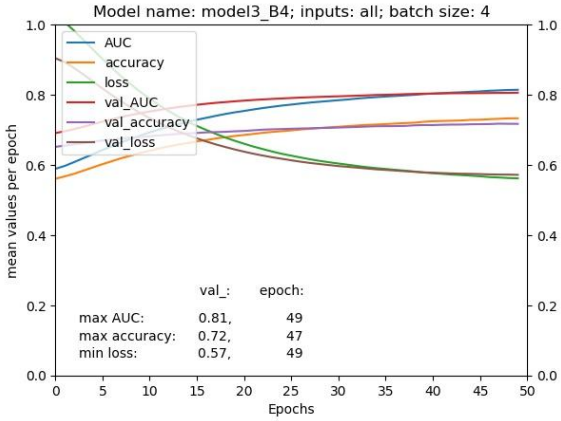
Model0, batch size equal to 32.



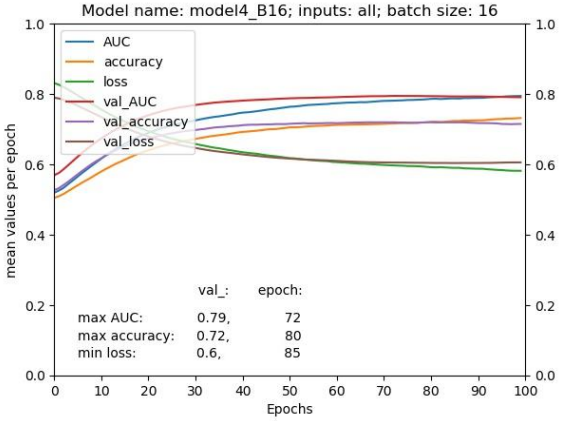
Model1, batch size equal to 32.



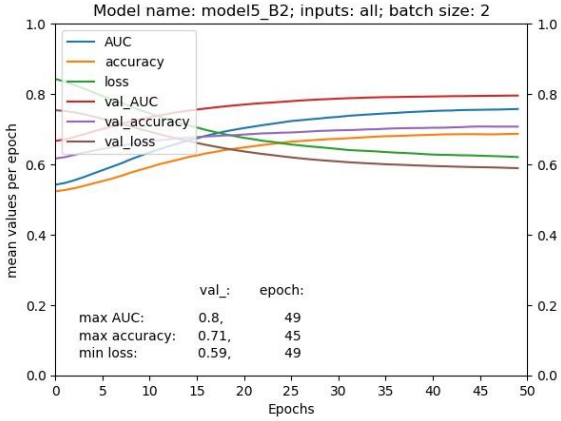
Model2, batch size equal to 4.



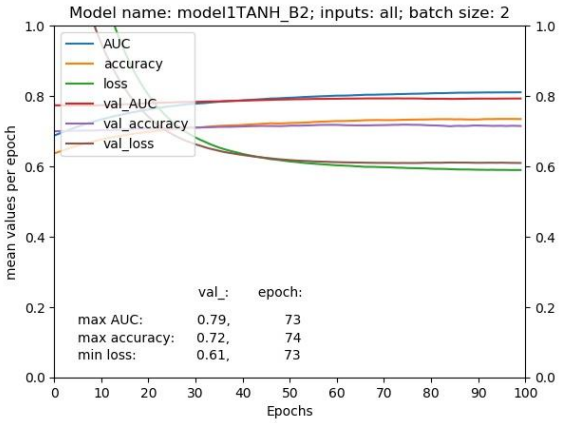
Model3, batch size equal to 4.



Model4, batch size equal to 16.



Model5, batch size equal to 2.



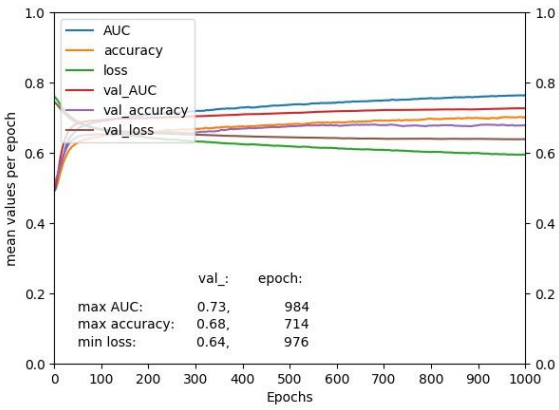
Model1, with *tanh* activation functions, batch size equal to 2.

We can see how, according to *loss* values, these systems seem to struggle to learn, and the *accuracy* and *AUC* values achieved are not as high as expected, as, maximum values reached are 73% and 0.81 respectively.

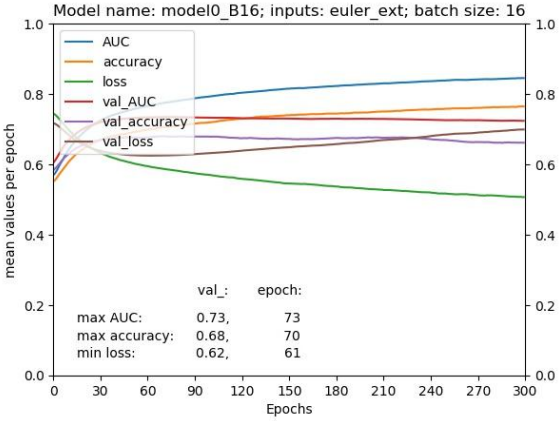
Besides that, the overfitting phenomenon seems to be determining in none but the model1, which was forced to that behavior.

Reduced-inputs-based

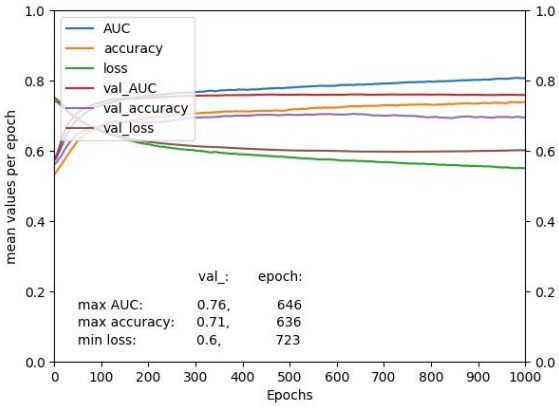
In addition to those models trained upon all possible numerical and categorical data, recall that, as I explained in section 3.6, MLPs was also tested only on certain subsets of features.



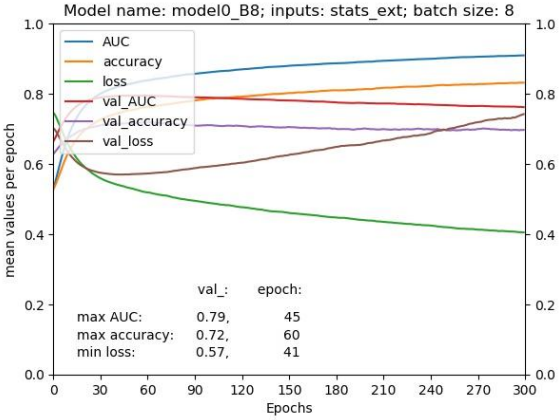
ECE output-based MLP score.



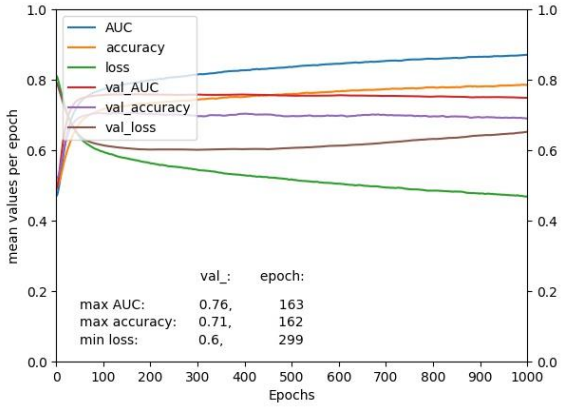
ECE and categorical info-based MLP score.



SFE output-based MLP score.



SFE and categorical info-based MLP score.



SFE and ECE-based MLP score.

These results were obtained for mostly *model0* and *model3*-like architectures. Again, the validation loss values suggest a poor learning, and even among the best

performances, the evaluation metrics values scored are not high above the random classification, achieving up to 72% accuracy in the best case.

4.1.2 CNNs

The best scores for CNNs models are summarized in the following table.

	AUC	accuracy
DenseNet210	0.78	0.70
MobileNetV2	0.74	0.67
EfficientNetB0	0.84	0.76

As you can see, the *EfficientNet*-based model got the best score. However, the validation loss didn't manage to lower very much:

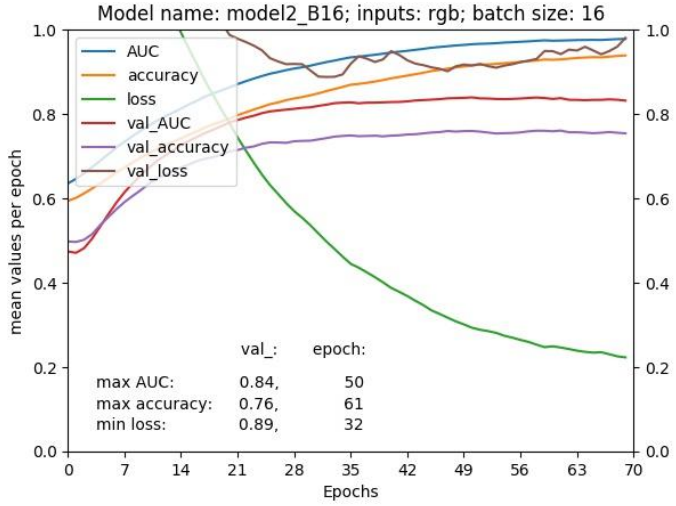


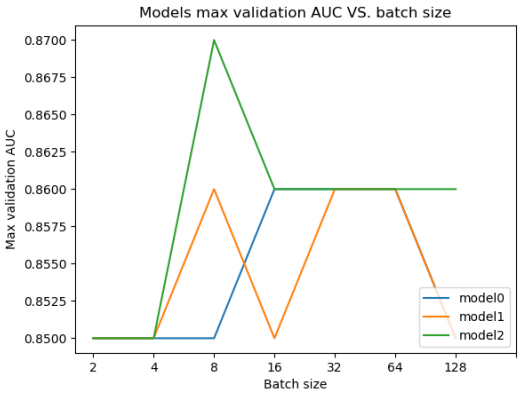
Figure 4.8: *EfficientNet*-based CNN score.

making us guess there would be architectures that could overfit later than that system. Other models' validation loss scores were by far higher than this one, so they were probably not controlling the overfitting at all, which can be sensible, for DenseNet, but not for MobileNet, given the size of each transferred model: ~ 20, ~ 4 and ~ 5 millions of parameters, for *DenseNet*, *MobileNet* and *EfficientNet*, respectively. That makes me think the topology of EfficientNet is simply more suitable for the problem under consideration.

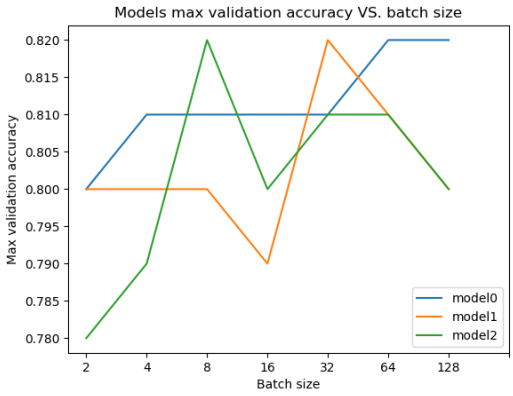
4.1.3 Hybrid Model

Recall: the hybrid model was the system that, being an MLP, added to the complete inputs sets another column, given by the CNN systems predictions over images.

The models that obtained the best results were:



(a) Mean AUC score vs. batch size.



(b) Mean accuracy score vs. batch size.

The score obtained by the hybrid model was considerably higher than all previous submodels, especially for the accuracy. Since the holdout validation technique has been used here, we cannot consider only the best score, as before, but instead the mean of all values must be computed in order to reduce the influence of a possible favorable dataset splitting.

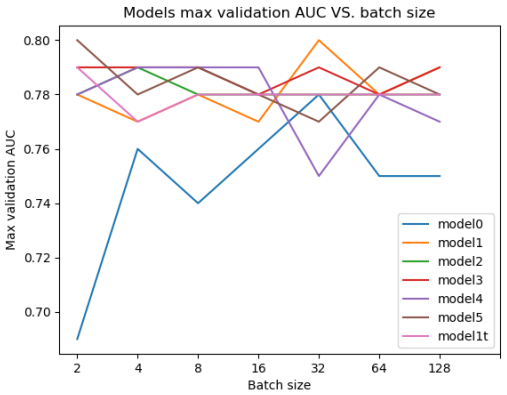
Among them, the *model2*, which was a mid-point between the small-sized *model0* and the large-sized *model1*, seems to have scored both the best AUC (0.87) and accuracy (82%), being for the latter tied with both others though.

4.2 Comparison

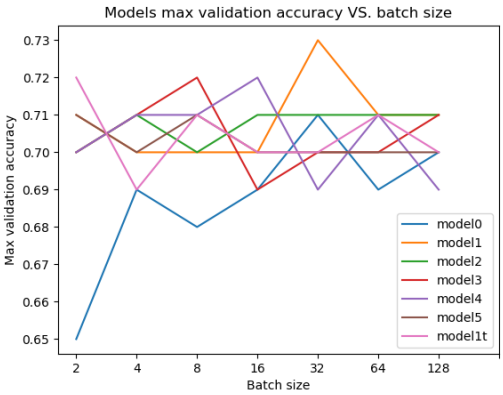
In this section I will *deductively* analyze the results, exposing the numbers obtained and the comparison among all of them, but without introducing any hypothesis about what is causing such performances or preventing them from getting higher scores; that will come later in the discussion.

4.2.1 Complete inputs MLPs

Since ANN are stochastic systems whose weights are initialized randomly, it's not so important to look only at *the best result* for each model, but instead it is interesting to do some statistics with all obtained scores. In the following figure you can see the AUC and accuracy scores respectively, for each chosen model.



(a) Models AUC score vs. batch size.



(b) Models accuracy score vs. batch size.

The *model1*, that was proposed in order to force the theoretical overfitting, seems to reach the best scores instead. For this reason, I decided to keep pushing the limit of overfitting, trying to achieve it by both bigger networks and experimenting from the *model1* architecture, with hyper-parameters, regularization techniques, and different activation functions, but without substantially changing the layers size and network's depth.

As expected, networks started overfitting quite soon, even for higher l_1 and l_2 regularization parameters, several dropout layers, and bigger batch sizes:

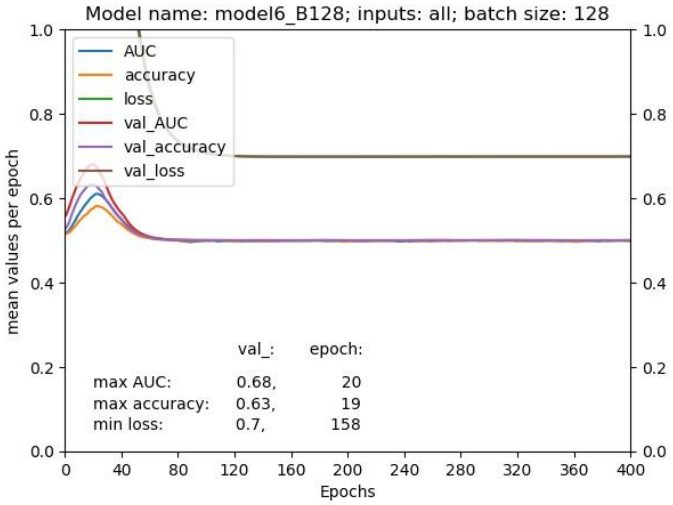


Figure 4.11: Overfitting

4.2.2 All inputs vs. reduced inputs MLPs

The following table summarizes all scores. The "-ext" suffix indicates the addition of categorical data (*sex* and *anatomical site*).

	AUC	accuracy
ECE	0.73	68%
SFE	0.76	71%
ECE-ext	0.73	68%
SFE-ext	0.79	72%
ECE-SFE	0.76	71%
Complete	0.81	72%

The reduced-inputs-based MLPs didn't manage to get as high scores as the complete-inputs-based one, but they weren't far either, scoring up to 0.76 AUC and 71% accuracy. As expected, less input info lead to worse system performance, but we can now notice that:

- The SFE process seems to be much more correlated to the target than ECE.
- The ECE algorithm seems to have provided a poorer improvement than the two categorical features to the SFE alone performance. Actually, ECE seems to have not helped at all to SFE since its presence makes no difference.
- Categorical data shows a significant improvement in AUC, while only 1% in accuracy.

4.2.3 MLPs vs. CNNs

The best scores achieved were:

	AUC	accuracy
MLPs	0.81	72%
CNNs	0.84	76%

Which means that the CNN learning directly from the original images seems to work better than the MLPs, that required instead a lot of DIP preprocessing steps and features extraction time investment.

I am not claiming the CNN to be *effortless*. Recall that, in order to produce the final CNN models and in addition to techniques also used in MLPs, we had to make use of:

- The transfer learning technique, which reuse an already trained system to other problems. Those systems are not quite trivial or small-sized; instead, they have several millions of parameters, and they require a huge computational cost to be trained.
- The data augmentation pipeline, which enriched the original dataset with modifications on images such as flipping, shearing, zooming, etc.

The complexity of the CNN systems is even higher than MLP models. What I am trying to point out is that, from a productivity and data-scientist time consumption perspective, the CNN seems to be by far the best option, between both considered.

4.2.4 Hybrid models

The best scores were obtained by the *model2*, and surprisingly for the same batch size, which is 8. But is it fair to claim that we obtained a system that scored so high? Well, recall that for hybrid models the holdout technique was used, which, even with the statistical correction of repeating the experiment for four different dataset division, is less reliable than the k-fold validation.

Is it fair then to claim the *model2* was the best? I don't think so. Surely it is what the experiment says, but I guess we should better keep safe and say that the best *three models* obtained an AUC ~ 0.86 and accuracy $\sim 81\%$, for batch sizes from 8 to 64, approximately.

4.2.5 Hybrid models vs. submodels

	AUC	accuracy
MLPs	0.81	72%
CNNs	0.84	76%
Hybrid	~ 0.86	$\sim 81\%$

The hybrid model managed to achieve higher scores, thanks to the union of information from different origins. Maybe CNN alone overtook the images features based MLP, but it is the synergy what really happens to get the best results.

Let me emphasize how promising is to us that the simple combination of subsystems that almost didn't passed the 75% accuracy, even for the small-sized dataset considered, made the accuracy go from 72%, 76% to 81%.

Especially in healthcare domain, may it be melanoma, breast cancer or COVID-19, that 9%-6% can make an extremely powerful difference. Given that improvement, there is now no doubt about how worthy it can be to keep investing time and computational effort in improving the pipeline, adapt the k-fold approximation, or searching for new hybrid options.

4.3 Discussion

In this section I will *abductively* analyze the results, trying to guess and explain why they are like they are.

4.3.1 About the Systems' Performance

MLPs

The MLP systems' accuracy score is surely not as high as one might expect for a reliable medical system. The best MLP scored an AUC and accuracy of 0.81 and 72% respectively, which can seem to be not that bad, given the experimental soul of the extraction of images features (recall that we have no clue about how accurate they are), but that good either, since what we are considering is a healthcare problem.

In the comparison section, ECE features is blamed for improving nothing in SFE alone performance. Two reasons may be causing this behavior:

- Each or some of the SFE features are more physically correlated to the malignancy of the skin lesion
- SFE features do not provide individually influence, but the whole complex do, as they are eleven features per image, while ECE only extracted five.

While the first reason cannot be solved since it is inherent to the nature of the features or their extraction, the second could be adjusted with several weights that enhanced the influence of those five numbers on the MLP.

CNNs

What about the CNNs? They have not been contaminated from the possible negligence of images topological, shape and color features extraction, so their performance is only thanks to (or because of) the network structure.

The yet-not-much-better performance can be caused by:

- The system: the architecture, the model chosen for transfer learning, hyperparameters, etc..
- The dataset: 1168 images is too few images. Even with techniques such as data augmentation, class imbalance correction, they are definitely too scarce (see 5.2.1 for more).

Hybrid system

I strongly believe the improvement we bumped into for the hybrid system is due to the heterogeneity of the information the system is fed with.

But why does it work better if it receives the image and more features extracted from images? Isn't it *redundant*?

As I explained in 3.6, the redundancy of a particular real world feature into more than one computation variable, is traduced in ANN as a higher influence of that feature into the predictor system. The system will *pay more attention* to such features because it is represented *twice* (or more times).

Now, of course we are not feeding the hybrid system twice with the same variable, but, consider one physical property of the image, let's say the red channel intensity of the center of the mole: if that property is significantly relevant, it will be *detected, measured and fed to the system almost twice*: once in the CNN training and prediction, and once in the SFE process. Same for the rest of the colors channels, area of the skin lesion, topology and other complexes shape features we can not really and deeply understand.

4.3.2 Why low accuracy but high AUC

What does a low accuracy but high AUC scores suggest?

If you think about it, the initial dataset was made by $\sim$ 98.3% benign cases, so, a simple system that predicted always "benign", would score as high as 98.3% accuracy. But what would the AUC score be?

The answer is a very low number, the AUC score aim is to identify right those prediction displacement due to classes imbalance. It is a measure on how much *sensitivity* and *specificity* the system can achieve, being those concepts the metrics on how the system detected correctly each class:

$$Sensitivity = \frac{TP}{P}$$

$$Specificity = \frac{TN}{N}$$

Where TP, P, TN, N, are *true positives*, *positives*, *true negatives*, and *negatives*, respectively. So the AUC score must come along with the accuracy score in order for the system results to be believable and realistic. So much so that the scored used in Kaggle to rule this competition was the AUC score, not the accuracy.

Does it mean that the performance of my system is actually acceptable? No, it doesn't. A high AUC does not enhance the accuracy, it only assures that the base general classification problem is being learned properly, and no class imbalance-based workaround has been used to obtain the scored accuracy. In other words, the proposed system may perform a $\sim$ 81% correct classification, but with a $\sim$ 0.86 AUC, so that it can be considered a more solid classification and less suspicious than one that scored 95% accuracy but lower AUC.

Chapter 5

Conclusion & Future Works

In this section the conclusion of the project will be discussed, as well as a speculation on what the future related works may be.

5.1 Conclusion

An intensive work has been done to reach the project completion. From the insight in AI and DIP techniques, to the design, development and deep experimentation with different models, capable of exploiting different types of information, in an intent of searching the solution to a relevant issue in healthcare domain.

- Many different features have been extracted from original images by means of DIP tools, and
- many different AI techniques have been used upon those features and their combinations.

Besides that, a deep comparison and abduction about the results have been done, leading the project to some key thoughts about the general improvement it can provide to the science as a melanoma detector system.

Along with the two mentioned above, a wide set of new skills have been developed trough the project life:

- Python programming deepening, acquiring an especial familiarity with images and AI related packages.
- The development of new methods when the existent ones didn't satisfy the project needs, applying an engineering perspective for wisely decided when already existent tools were useful and when new ones needed to be manufactured.
- Progress in aspects such as python packages versions control, server usage by means of *ssh* and *scp* protocols, some concepts about the GPU programming.
- Application of novel techniques that were not covered in my engineering studies or in any other activity I have worked so far, such as CNN, transfer learning, data augmentation, etc.

However, to me this project is only a seed of a growing plant, that would have no meaning without a proper perspective on what is coming next. Metrics scored by the proposed systems weren't finally as encouraging as expected (although the difference between them was), so many doors are waiting for us to be opened; please read about them in the section below.

5.2 Future Works

Many futures works can be proposed; I am going to describe the ideas that sound to me the most promising ones.

5.2.1 General Improvement

The first interesting line of possible future work that comes to my mind is to improve the current project. As suggested above, the scores obtained were not reliable enough for a medical system. Quoting again [37]:

"To prevent a model from learning misleading or irrelevant patterns found in the training data, the best solution is to get more training data. A model trained on more data will naturally generalize better."

I think this is the fundamental idea that could enhance this project: obtaining more data. Since the malignant cases are much less likely than benign (fortunately), either the dataset size will always be limited to the malignant cases size doubled, or new class imbalance correction factors are developed, maybe by:

- Keep punishing differently the network depending on the class it mistook, which is the already adopted approach, but may work better for biggest training sets and stronger correction factors.
- Data augmentation pipelines depending on the class of the image; in other words, what if only the malignant cases were data-augmented?

5.2.2 Use Patient ID Knowledge

If you recall from the EDA, section 3.1, there was a particular column, I have *wisely* ignored: the *patient-id*.

According to [1], and quoting [44]:

"The Ugly Duckling is another warning sign of melanoma. This recognition strategy is based on the concept that most normal moles on your body resemble one another, while melanomas stand out like ugly ducklings in comparison. This highlights the importance of not just checking for irregularities, but also comparing any suspicious spot to surrounding moles to determine whether it looks different from its neighbors. These ugly duckling lesions or outlier lesions can be larger, smaller, lighter or darker, compared to surrounding moles. Also, isolated lesions without any surrounding moles for comparison are considered ugly ducklings."

There is a really important and different information in the *patient-id* column, as it is the number which relates all mole images of the same patient, and can be

helpful in identifying if any of those moles is a melanoma. So, why have I ignored that column? Well, firstly, because of the length and depth of the project. I wanted to experiment in many DIP techniques, and extract many features from images, instead of focusing on columns. Besides that, in order to make a good use of the patient-id column, images should have been selected manually, including those patients who have more than one related images; this is clearly in contrast with the policy adopted to perform the down-sampling, I couldn't have done much work in DIP without, which leads us again to the previous reason.

However, the patient identifier represents a new source of knowledge *of different nature*, that could therefore increment the accuracy of the system, one would speculate.

5.2.3 Different Final Hybrid Systems

Another opportunity is to create different hybrid MLP and CNN-based systems that would accomplish a final prediction. The chosen method was to add one column for the CNN prediction to the complete dataframe, but the final classification could also be achieved by several other ways:

- join both systems in a huge computation effort: join MLPs and CNN in a unique ANN Keras model,
- add one column for each CNN prediction to complete the dataframe (*i.e.* including also hairs-removed and segmented images-based CNNs predictions), and
- create a final dataframe with a prediction for each submodel (MLPs and CNNs), *i.e.* totaling up to seven predictions.

5.3 Final Personal Comments

To end with, I would like to express my personal opinion about how I undertook all the challenges the project raised.

The general mindset I adopted, as explained in the introduction, was of an experimental nature. I am a creative student who, rather than only exploring and studying the current science boundaries, prefers to *design new ones* on his own. This is the reason why you may have noticed a shortage in literature and state-of-art citation about the different techniques used. Please forgive me, it is in my essence, I tend to genuinely experiment by myself before reading anything else.

When I studied DIP and AI subjects during the degree I had a taste of how astoundingly powerful those tools were, and this idea planted in my mind: searching for new ways of joining them, new synergies, new mathematical backgrounds capable of handling and making the most of both. And using those new tools to solve healthcare problems, which I really feel a calling to.

Now, I still haven't got what is needed to be in the front line of the army of scientists that all over the world go hunting those new systems, but this project was to me a start point. I really had to work hard in order to make all of the different parts of systems match properly, and, to my surprise, the results were positive, as I finally could demonstrate the power of the hybrid system compared to the submodels, the power of the synergy and heterogeneity I strongly believe in, despite questioning the extent of the results obtained (they will definitely require further experimentation).

This project has given me the opportunity to conduct the largest experiment so far in my life, while being supervised by the supportive professor Luis Valencia Cabrera, to whom I express my gratitude again.

I hope you found it interesting in its complexity, while also gentle to follow. I did my best for it.

Bibliography

[1] Kaggle. *SIIM-ISIC Melanoma Classification*. URL: https://www.kaggle.com/c/siim-isic-melanoma-classification.

[2] David J. Griffiths. *Introduction to Electrodynamics*. 1981. ISBN: 0-13-805326-X.

[3] Skin Cancer Fundation. *Skin Cancer Facts and Statistics: What You Need to Know*. URL: https://www.skincancer.org/skin-cancer-information/skin-cancer-facts/.

[4] WebMD. *Skin Cancer*. URL: https://www.webmd.com/melanoma-skin-cancer/melanoma-guide/skin-cancer#1.

[5] World Cancer Research Fund. *Skin cancer statistics*. URL: https://www.wcrf.org/dietandcancer/cancer-trends/skin-cancer-statistics.

[6] Kaggle. URL: https://www.kaggle.com/.

[7] *Society for Imaging Informatics in Medicine*. URL: https://siim.org/.

[8] *International Skin Imaging Collaboration Archive*. URL: https://www.isic-archive.com/#!/topWithHeader/wideContentTop/main.

[9] DIP. *Grayscale to RGB Conversion*. URL: https://www.tutorialspoint.com/dip/grayscale_to_rgb_conversion.htm.

[10] CS courses. *Images as functions*. URL: https://courses.cs.washington.edu/courses/cse557/00wi/lectures/imageprocessing.pdf.

[11] Aishwarya Singh. *Demystifying the Mathematics Behind Convolutional Neural Networks (CNNs)*. URL: https://www.analyticsvidhya.com/blog/2020/02/mathematics-behind-convolutional-neural-network/?utm_source=blog&utm_medium=cnn-vs-rnn-vs-mlp-analyzing-3-types-of-neural-networks-in-deep-learning.

[12] Robert Fisher et al. *Digital Filters*. URL: https://homepages.inf.ed.ac.uk/rbf/HIPR2/filtops.htm.

[13] Ashley Walker Robert Fisher Simon Perkins and HYPERMEDIA IMAGE PROCESSING REFERENCE Erik Wolfart. *Morphology*. URL: https://homepages.inf.ed.ac.uk/rbf/HIPR2/morops.htm.

[14] Preechaya Srisombut. *Morphological Image Processing*. URL: http://graphics.ics.uci.edu/CS111/Slides/woodsandgonzalez.pdf.

[15] Brett Grossfeld. *Deep learning vs machine learning: a simple way to understand the difference*. URL: https://www.zendesk.es/blog/machine-learning-and-deep-learning/.

[16] Wings for life. *How does a neuron work?* URL: https://www.wingsforlife.com/en/latest/how-does-a-neuron-work-562/.

[17] DataFlair Team. *Artificial Neural Networks for Machine Learning – Every aspect you need to know about.* URL: https://data-flair.training/blogs/artificial-neural-networks-for-machine-learning/.

[18] Computation Science and Artificial Intelligence Department of the University of Seville. *Notes on Neural Networks.* URL: http://www.cs.us.es/cursos/siis/temas/tema-06.pdf.

[19] Aravind Pai. *CNN vs. RNN vs. ANN – Analyzing 3 Types of Neural Networks in Deep Learning.* URL: https://www.analyticsvidhya.com/blog/2020/02/cnn-vs-rnn-vs-mlp-analyzing-3-types-of-neural-networks-in-deep-learning/.

[20] Trigram. *SIIM: d3 EDA, Augmentations and ResNeXt.* URL: https://www.kaggle.com/nxrprime/siim-d3-eda-augmentations-and-resnext.

[21] Dongju Liu and Jian Yu. "Otsu Method and K-means". In: *2009 Ninth International Conference on Hybrid Intelligent Systems* (2009). DOI: 10.1109/HIS.2009.74.

[22] Halil Murat Ünver and Enes Ayan. "Skin Lesion Segmentation in Dermoscopic Images with Combination of YOLO and GrabCut Algorithm". In: *Diagnostics (Basel). 2019;9(3):72.* (2019). DOI: 10.3390/diagnostics9030072.

[23] Willi A. Kalender. *Computed Tomography.* ISBN: 978-3-89578-317-3.

[24] By Ihab Zaqout. "An Efficient Block-Based Algorithm for Hair Removal in Dermoscopic Images". In: *Computer Methods and Programs in Biomedical Signal and Image Processing* (2017). DOI: http://dx.doi.org/10.5772/intechopen.80024. URL: https://www.intechopen.com/books/computer-methods-and-programs-in-biomedical-signal-and-image-processing/an-efficient-block-based-algorithm-for-hair-removal-in-dermoscopic-images.

[25] Pedro Bibiloni. *Hair Removal in Dermoscopic Images with Soft Color Morphology.* URL: https://github.com/PBibiloni/hair-removal.

[26] sunnyshah2894. *DigitalHairRemoval.* URL: https://github.com/sunnyshah2894/DigitalHairRemoval.

[27] Teresa Mendonça et al. "Comparison of Segmentation Methods for Automatic Diagnosis of Dermoscopic Images". In: *Proceedings of the 29th Annual International Conference of the IEEE EMBS* (2007).

[28] Tony Chan and Luminita Vese. "An Active Contour Model without Edges". In: *Scale-Space Theories in Computer Vision* (1999). DOI: 10.1007/3-540-48236-9_13.

[29] Michael Kass, Andrew Witkin, and Demetri Terzopoulos. "Snakes: Active contour models". In: *International Journal of Computer Vision volume 1, p. 321–331* (1988). DOI: 10.1007/BF00133570.

[30] Scientific Figure on ResearchGate. *Segmentation of Soft atherosclerotic plaques using active contour models*. URL: https://www.researchgate.net/figure/Segmentation-based-on-active-contours-Green-Initial-contours-Red-Final-Contours_fig8_305787165.

[31] Pascal Getreuer. "Chan-Vese Segmentation". In: *Image Processing On Line, 2, pp. 214-224* (2012). DOI: DOI:10.5201/ipol.2012.g-cv.

[32] Rami Cohen. "The Chan-Vese Algorithm". In: *Project Report* (2011). DOI: arXiv:1107.2782.

[33] Skimage. *Chan-Vese Segmentation*. URL: https://scikit-image.org/docs/dev/auto_examples/segmentation/plot_chan_vese.html.

[34] Wikipedia. *Topology*. URL: https://en.wikipedia.org/wiki/Topology.

[35] A. Giuliano Mirabella. *Topological Analysis of nD images*. URL: https://github.com/agiulianomirabella/tani.

[36] A. Giuliano Mirabella. *n-Cells Identification Model*. URL: https://github.com/agiulianomirabella/tani/blob/master/n_Cells_Identification_Model.pdf.

[37] François Chollet. *Deep Learning With Python*. ISBN: 978-1-61729-443-3.

[38] George Novack. *Building a One Hot Encoding Layer with TensorFlow*. URL: https://towardsdatascience.com/building-a-one-hot-encoding-layer-with-tensorflow-f907d686bf39.

[39] Keras. *Model training APIs*. URL: https://keras.io/api/models/model_training_apis/.

[40] Pinterest. *Training from scratch vs Transfer learning*. URL: https://pl.pinterest.com/pin/672232681856247783/.

[41] Keras. *Applications*. URL: https://keras.io/api/applications/.

[42] Sarang Narkhede. *Understanding AUC - ROC Curve*. URL: https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5.

[43] A. Giuliano Mirabella. *AI and Digital Images Processing Techniques in Melanoma Detection*. URL: https://github.com/agiulianomirabella/melanoma-detector.

[44] Skin Cancer Fundation. *Melanoma Warning Signs*. URL: https://www.skincancer.org/skin-cancer-information/melanoma/melanoma-warning-signs-and-images/.