AI and Digital Images Processing Techniques in Melanoma Detection

A. Giuliano Mirabella Galvín

September 2020

Abstract

The aim of this project is to create an intelligent system capable of detecting melanoma in skin lesion images. Taking advantage of the “SIIM-ISIC Melanoma Classification" dataset on Kaggle [11], I will use an hybrid Artificial Intelligence and Digital Images Processing based approach to design algorithms to correctly classify skin images into benign or malign skin lesion.

Acknowledgements

I want to thank...

Table of Contents

Chapter 1

Introduction

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

In this project I will show how far technology can go in automatic diagnostic. I will join both Image Processing 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 you will read about the health problem to solve, as well as 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 the skin cancer to be the most prevalent type of cancer around the world [7, 11, 31], others -such as the World Health Organization- place it *only* in the fifth position.

Whichever statistic may be right, we can safely assume the skin cancer to be one of the most important health problems to solve in nowadays medicine. According to SCF, here are some stats & facts 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 these statistics, as according to the American Institute for Cancer Research [6], USA is *only* in the 17^{th} position in Countries with highest rates of melanoma list -preceded by many European ones, by the way.

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 you will 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 skin cancer, it is by far the biggest threat, causing the 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, 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 [11], a public competition in Kaggle.

Kaggle [10] is an online community of data scientists and machine learning practitioners. 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) [25] is 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.

SIIM is joined by the International Skin Imaging Collaboration (ISIC) [9], 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.

1.4 The Project

This project has to be understood as an experiment. The fundamental idea is to apply AI techniques both on the original images and on certain images numerical features extracted by means of digital images processing (DIP), and than compare the results obtained. One important detail to take into account for the latter though is that all along the project, I will have no 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*.

In this project, I will (hopefully) give you in chapter 2 a general understanding about the fundamentals of both used technologies, than, in 3 I will describe carefully how the images features are going to be extracted, and finally I will expose in 4 the results and conclusions of the project.

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."

Chapter 2

General Overview of the Involved Technologies

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). For each one of them, several concepts will be described to show some examples of their usefulness.

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 a subcategory of digital signal processing where the signal has two important properties:

- it is usually 2 or 3 dimensional (2D or 3D images), 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 you should by now think of as a 2D matrix containing a 3D vector in each node expressing the red, green, and blue value (RGB) that node take 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 entirely be written in Python language, so it is worth to explain how matrices can be handled. With the help of *numpy*, a package for nD array treatment, the management of tensors is made very simple and intuitive, as if the were scalar numbers. Besides that, the most used package in the project will be *skimage*, a module with plenty of useful image processing built-in functions.

Below, I am going to formally define some of the simplest operations executable to digital images, in order to strengthen the idea of the matrix nature of images. When we assume the image is grayscaled, we do it for simplicity, the functions definitions are the same for RGB images.

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 be 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 into a new size ($a' \times b' \times ... \times z'$) would be a new image $Z = z_{ij..k}$:

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

As explained above, an RGB image is the same of grayscaled, but each element has three components; so how is an RGB image X converted to a grayscaled $Z = z_{ij..k}$?

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

being $x^y_{ij...k}$ the y color component of the $ij...k$. 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 [5].

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 x m) image X and a (a x b) kernel f, returns a matrix formed by the result of element-wise multiplication between the kernel and certain possible (a x b) "windows" of X.

The formal definition of a convolution of a kernel f over a 2D image X is given by (from [4]):

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

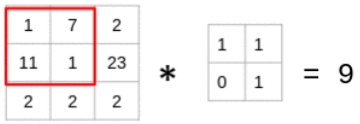
Let's visualize it with an example, taken from [23]. Let the image to convolve:

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

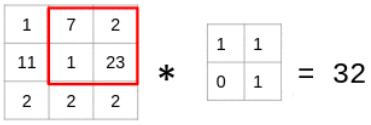
And the kernel:

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

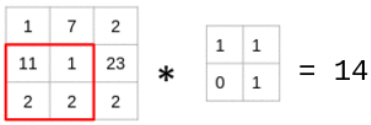
Graphically, the process of convolving is:



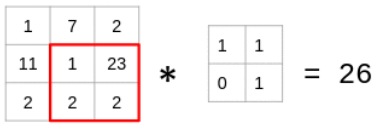
(a) 1^{st} window



(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 whatever size:

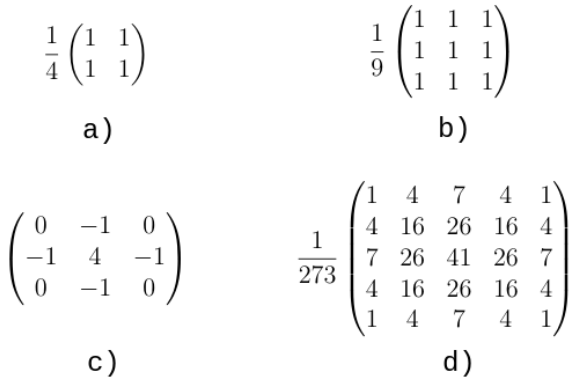


Figure 2.2: a) 2x2 mean filter. b) 3x3 mean filter. 3x3 laplacian filter. 5x5 gaussian filter.

See [20] for more info about digital filters.

2.1.3 Morphological Operations

Morphological operations are also convolution-based. 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 neighbor pixels, where the neighborhood is defined by the structuring element.

There are four main morphological functions: *erosion*, *dilatation*, *opening* and *closing*, [21].

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, then erosion is given by

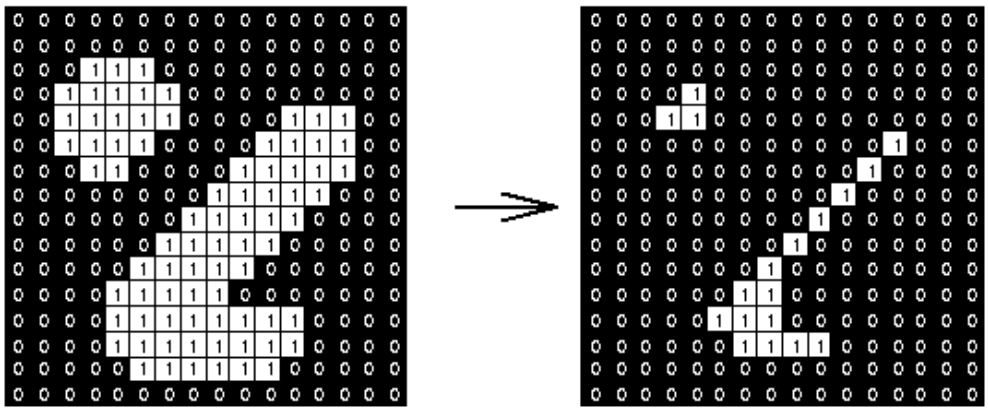


Figure 2.3: erosion of an image by a 3x3 isotropic structuring element

You can think of it 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:

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

Graphically:

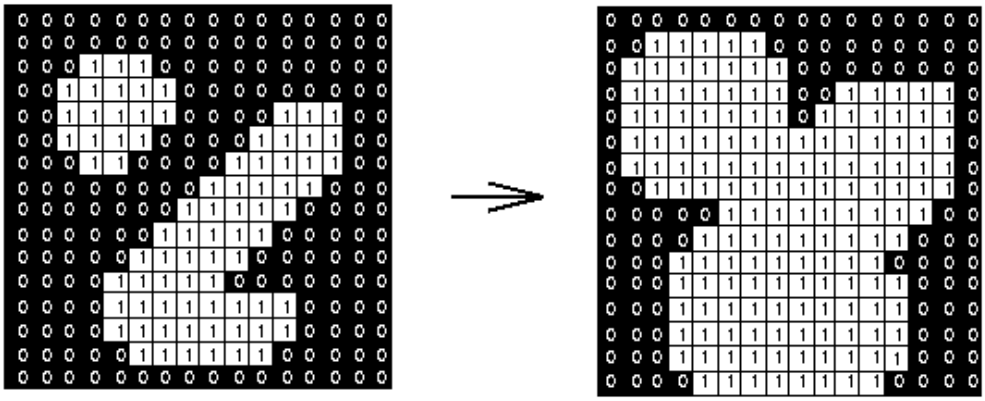


Figure 2.4: dilatation of an image by a 3x3 isotropic structuring element

Again, you can think of it 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:

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

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:

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

i.e. it is the erosion of the dilatation. Used for suppress small enough white components instead. Opening and closing are idempotent operations and among their many applications there are:

Salt-and-pepper noise correction: salt-and-pepper noise is a form of noise sometimes seen on images that presents 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; here is an example from [26]:

(a) salt noise

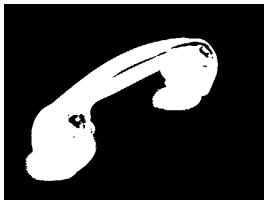
(b) Opened image

(c) pepper noise

(d) closed image

Figure 2.5: salt and pepper noise correction

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



(a) original image

(b) closed image

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 the design of systems that simulate what us humans understand of intelligence. The concept of intelligence is probably one of the most unknown thing 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-technology I am going to use is Deep learning.

2.2.1 Deep Learning

Deep Learning is the evolution of Machine learning.

Machine learning consists of "algorithms that parse data, learn from that data, and then apply what they have learned to make informed decisions" [8]. A machine learning system is an automatic application that, *with the help of a data engineer*, who design and code proper statistical models, can make predictions about some variable, but can not figure out whether that prediction was correct or not, neither update automatically itself to predict better next time.

Deep learning is an evolution of Machine learning where the system does not need any more a previous hand-designed statistical model's rules to work; instead, the system is capable of interacting with the target variable to predict, compare it with its predictions, and learn by making mistakes, all by itself.

The strongest tool of deep learning is Artificial Neural Networks (ANN) and we will use it in two flavors: Multi-Layer Perceptron (MLP) and Convolutional Neural Network (CNN). An ANN is a human-brain-inspired system formed by a set of artificial neurons-cells-inspired units called perceptrons.

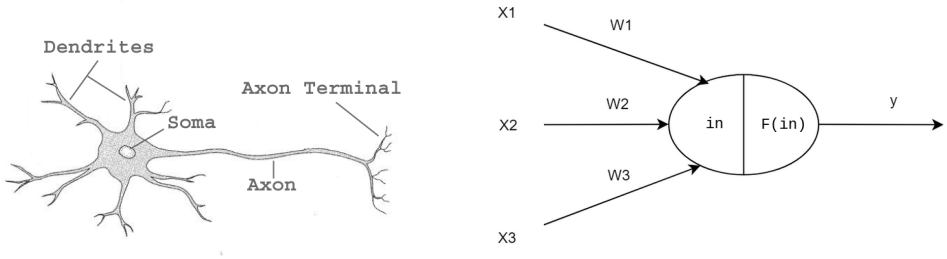
2.2.2 The Perceptron

The basic structural unit of ANN is an artificial neuron or a perceptron. Perceptrons interconnects with each other receiving, processing and emitting signals to others, thus finally forming he 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 is received, the cell body (what contains the soma in the figure),

where the signal is processed and redirected, and the axon, that emits the stimulation to others neurons' dendrites through structures named synapses [15].

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



(a) Our neurons (b) Artificial neuron

Figure 2.7: real neurons vs. artificial ones

Inspired by the real one, the artificial 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 form the final input $in = \sum_{j=1}^{n} w_{ij} x_j$,
- a *processing* F of the input and
- an *output* y that will hopefully arrive to another neuron.

Now, the perceptron output behavior can be modified as one like 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, or remain inactive, emitting $y = 0$. This behavior is fulfilled by:

- $x_0 = -1$, it works as an activation threshold
- the function F, generally named "activation function" is generally 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:

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

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

2.2.3 MLP

A MLP is a mathematical model based in a graph structure whose nodes are artificial neurons (figure 2.8).

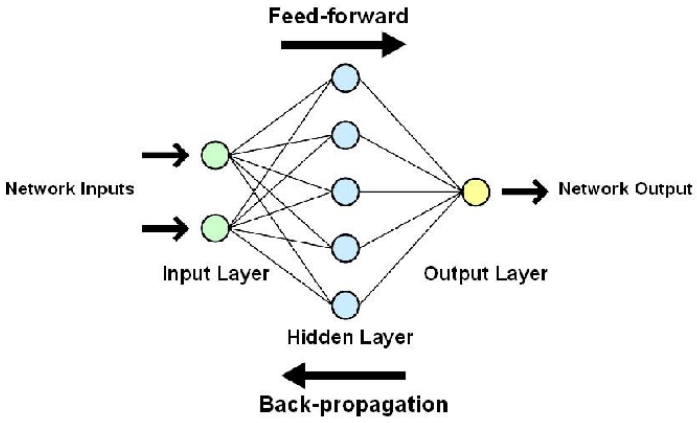


Figure 2.8: A simple MLP

Every node is connected to others trough 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 out of the inputs it receives and works likewise as an input for others nodes [22].

As you can see in the figure 2.8, similarly to a single perceptron, a 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. A 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, thus it can be used as a classifier in $\mathbb{R}^n$ into m classes.

So, given this graph-based structure, how does a 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 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 consists of performing a reward-punishment formula over certain inputs the correct labels are already known. For this purpose:

- split that dataset into *training set* and *test set*,
- make the network predict labels on *training set*,
- "reward" the network if its prediction is correct and "punish" it otherwise and
- test the network performance on the *test set*.

Initially, the network speculates rather than predict, since it has no clue about what the relation between inputs and outputs is supposed to be. It is not before the network is trained that first reasonably correct results come

out.

If you think of it, the baby from the before 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 saying simply "not a cat, it's a dog!". The MLP are just oversimplified version of our brains, and they need to *learn* the same way.

But how do we reward or punish our network? You may have notice there is another flow of information in figure 2.8, apart from the forward communication path above explained: 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 neuron once the reward or punishment comes from the output layer. The algorithm is built upon the gradient descent idea: it simply drags the error (or the success) result of comparing the prediction with the known correct output on the output layer backward to the hidden layers up to the input one, lowering (or enhancing) the weights of the connections that activated for that wrong (or correct) prediction. You can visit the appendix 5.1 for a more formal description.

2.2.4 CNN

Could an MLP accept an image, which is a 2-dimensional matrix as an input? The answer is yes. Pixels values could be flatten to a 1-dimensional vector an can be processed such as any other input, but there is a better way to do it. When it comes to flatten the image pixels values, a special property is being 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 special properties, such as edges and geometrical structures, in both high or low level of cognition.

Convolving an image results in a feature map (see figure 2.9, from [19]).

Figure 2.9: 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 reward is given to a prediction, every connection used for that prediction is strengthened, and vice versa, (figure 2.10).

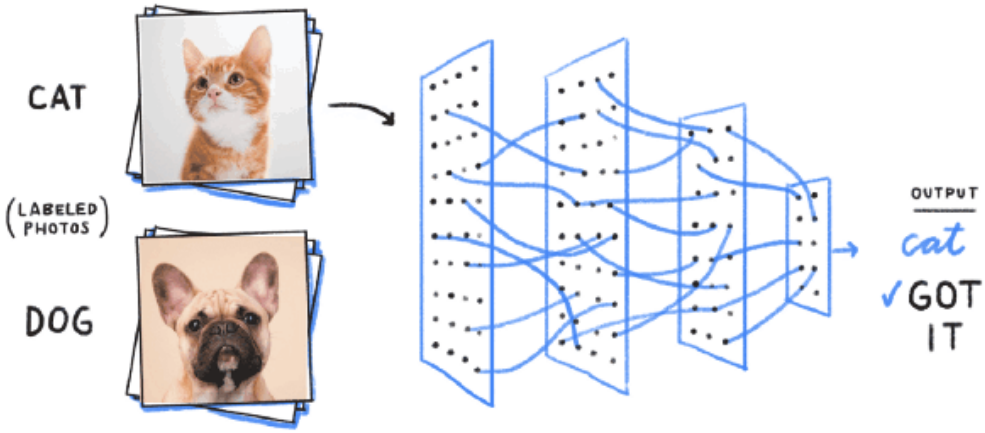


Figure 2.10: A correct classification strengthen the blue connections

Thus achieving a continuous updating of kernels weights, which ultimately leads to a more accurate features extraction.

So far, a general introduction to DIP and AI worlds has been done, from

the mathematics fundamentals, through to their simple intuition as a real-world inspired tech tool, up to several of their techniques and systems, 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 malign melanoma from benign?

In this chapter we 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 data describe trough 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 format, 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

Below, we are going to visualize the distribution of each column versus the target variable. First of all, let's visualize the *.csv* 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 info

So there are 33126 rows and 8 columns. 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 [29], 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:

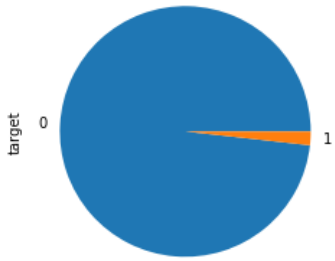


Figure 3.2: dataframe info

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

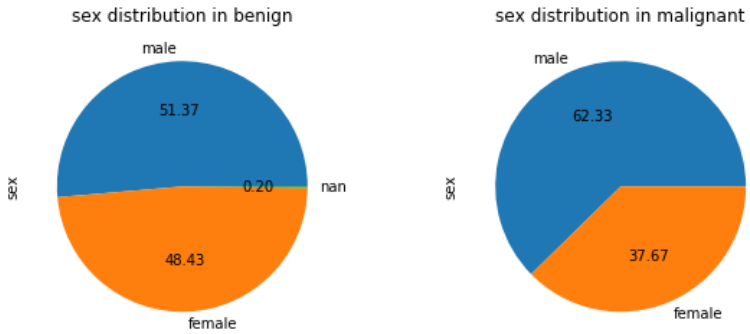


Figure 3.3: dataframe info

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:

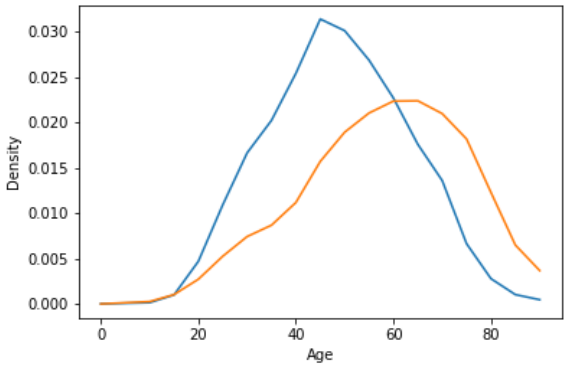


Figure 3.4: dataframe info

Even if the mean is a little bit displaced to the right, it seems to be well distributed. Now it's the turn for the *anatom_site*:

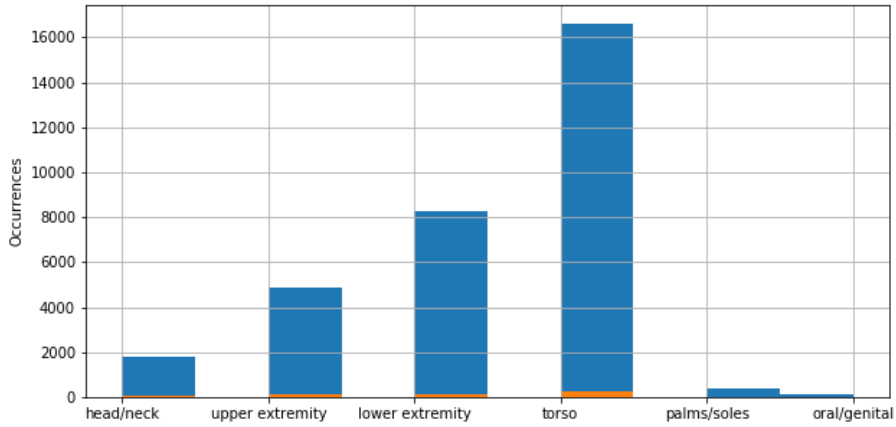


Figure 3.5: dataframe info

And the *diagnosis*:

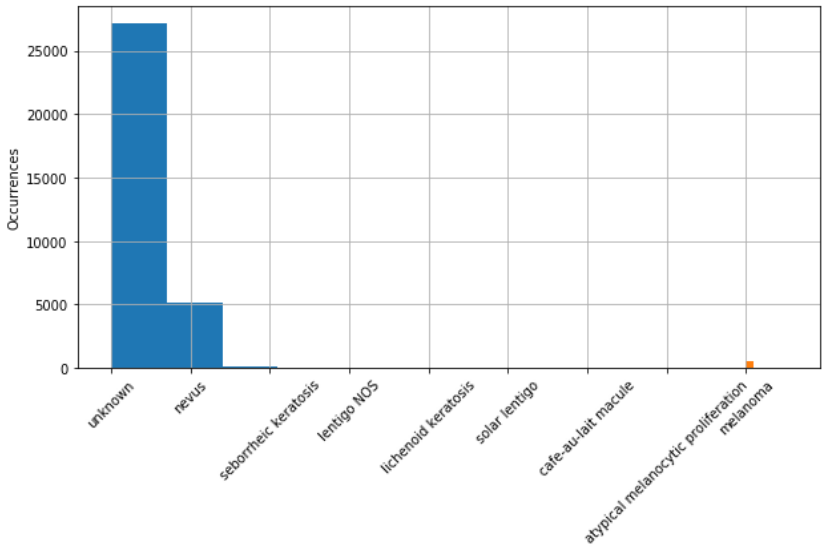


Figure 3.6: dataframe info

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 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:

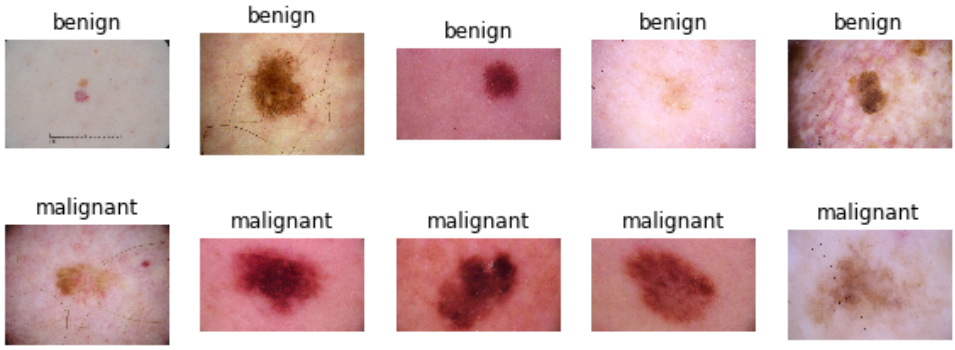


Figure 3.8: Some random images

3.1.1 Dataset Strategy

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*.

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

Down-sampling We have seen how strongly the target variable is imbalanced. Only 1.63% of the cases are malignant, which is an unacceptable dataset equilibrium to train any intelligent system on.

Furthermore, recall this is an hybrid project, made on the purpose of joining the DIP and AI worlds together. The images processing along the pipeline are not quite little computationally expensive, 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. The dataset is going to be reduced to the first 5000 cases plus the exceeding malignant ones, totaling 5509 cases, being 4925 benign and 584 malignant, 89.4% and 10.6%, respectively.

3.2 The Images Features Extraction Pipeline

Now, it is time to state what the flow of information, from the initial pixels values of the starting images dataset to the final dataframes obtained is, what processes the images are going to experience.

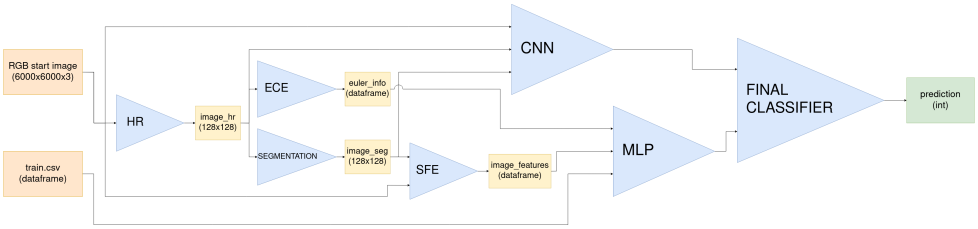


Figure 3.9: Data Pipeline

In the above diagram,

- 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

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.

3.3 Segmentation

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

Segmentation of the object X consists of binary classify 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

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 a 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.. oh, and his face too!

This happens because of the method used to segment that image, which in this case happen to be a threshold function assigning the value 1 to those pixels with higher than a threshold t values 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 is the Otsu's method, it is chosen to minimize the 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 say 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 join DIP and become nowadays our best chance to accomplish this task, making computer vision application each day better and better at recognizing objects in images and videos.

To me, is really intriguing how far DIP algorithms alone are from accomplish 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 much more different from us than 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 this (from [30]):

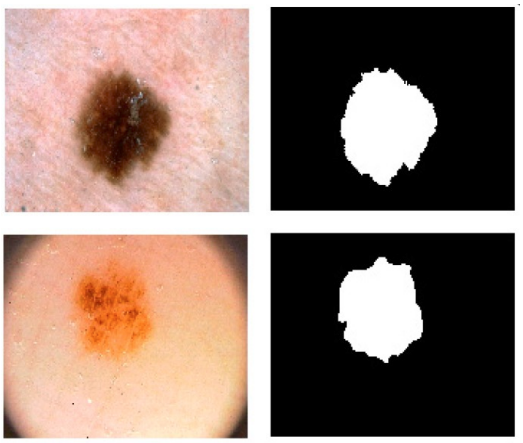


Figure 3.11: Skin lesion segmentation ideal output

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 [12]. Hairs introduce new edges and regions in the image that can mislead the segmentation algorithms into wrong conclusions.

For instance, 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 in an initial contour that is deformed until it hopefully land into the foreground-background edge, thus is perfect for analyzing the influence of hairs:

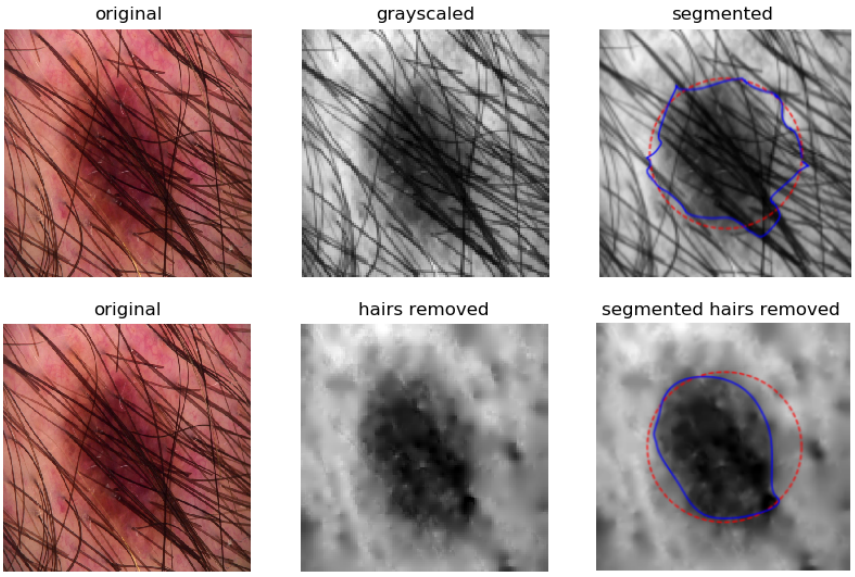


Figure 3.12: original image segmentation vs. hairs-removed image segmentation

Above in segmented images, the red circumference is the initial contour to be deformed, and the blue curve is the final one, 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 get stuck in those edges, thus producing a wrong output and misidentifying the skin lesion boundary.

Once stated a hair removal process (HR) needs to be done 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

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

The fundamental of this algorithm is to compute the result of the image minus the *closing* of itself. Let's examine what that means.

Following the article [33], given a grayscaled image, the algorithm steps are:

YIQ color space conversion: YIQ ((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 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 [33]):

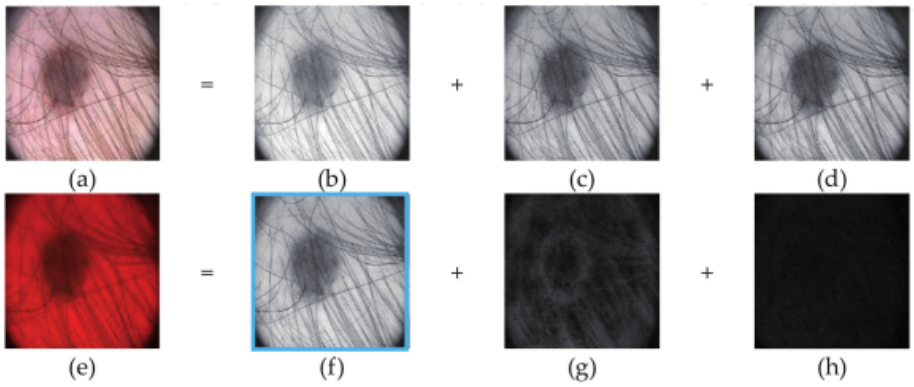


Figure 3.13: A digital dermoscopic image presented in RGB (a-d) and YIQ (e-h) color spaces

It is easy to see how the Y channel gives to hairs much stronger emphasis than I and Q do, while they are present in every RGB channel. This show 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:

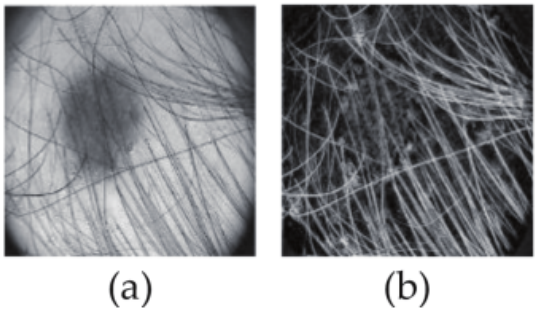


Figure 3.14: The ideal hairs detection result. (a) Y-channel image. (b) Result of bottom-hat operation

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

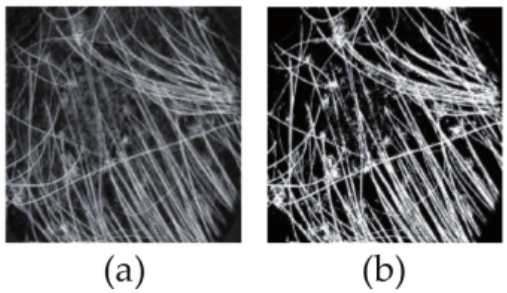


Figure 3.15: Hair mask. (a) Result of bottom-hat operation. (b) Result of Otsu's method

Image inpainting: Image inpainting 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 neighbors values and making the transitions as smooth as possible.

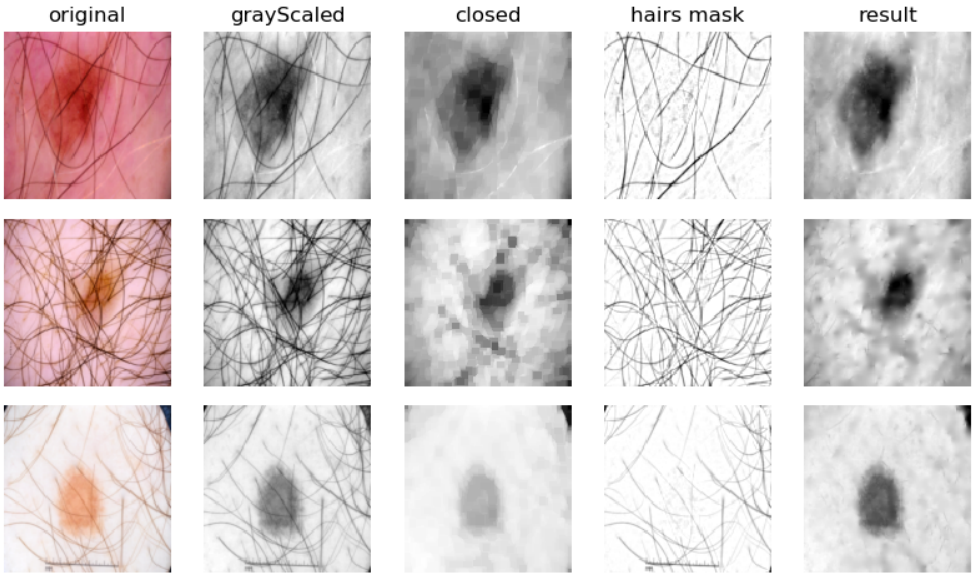
Implementation

For implementing the hairs removal algorithm, two attempts have been made to make use of already defined tools from gitHub platform, [1, 27]. However, for my particular case, both of them didn't manage to work out proper hairs mask, so I decide 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 [28].
- the hairs mask is black-dilated before the inpainting, so that it can cover properly the hairs regions.

In the following figures, the performance of my HR algorithm on manually selected images is going to be 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 a 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, compare graphically what the performance without and with the HR is, using the Chan-Vese algorithm (described in section 3.3.3).

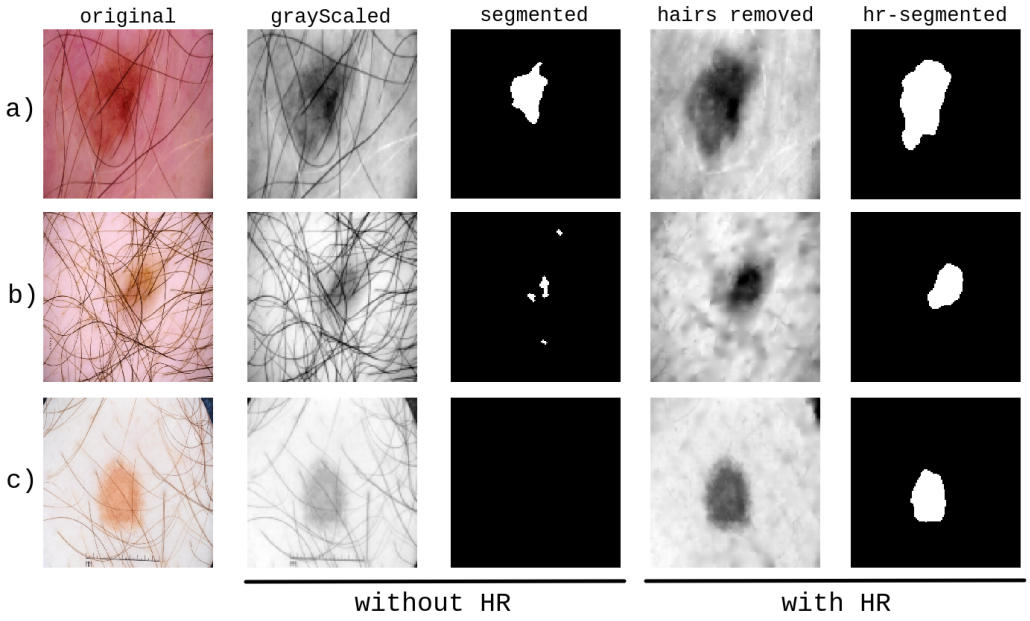


Figure 3.16: a) Eventually, the segmentation output was simply not complete without the hairs removal step. b) Sometimes, the hairs really misled the algorithm. c) Others, it was not even detected any spot at all, while with the improvement of hairs removal it is.

In figure 3.16 the segmentation improvement of a hairs removal preprocessing is shown.

3.3.2 Active Contours

The Chan-Vese Algorithm (CVA) was proposed in 1999 as "An Active Contours Model Without Edges" [2], i.e., as an improvement of an already existent algorithm called *active contours*. Let's first talk below about it.

The active contours model, also named "snake", is introduced in 1988 [13] 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 of 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 $s \in [0, 1]$, than 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, it depends only on C, and gives a score about how elastic and smoothed the snake is, information encapsulated in the first and second derivative of C:

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

  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 actually they need to be changed to finite series.

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.

Here you can see how well it can fit a non geometrical shaped object:

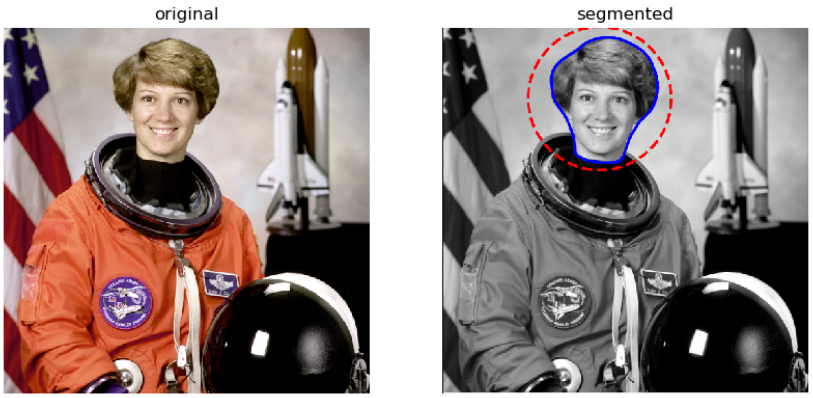


Figure 3.17: ACM segmentation of the astronaut face; the red and blue curves are the initial an final snakes, respectively

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 the figure 3.17 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. 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.

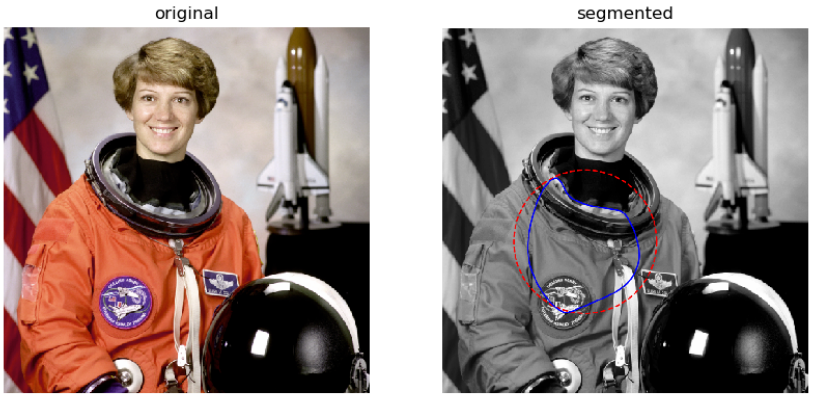


Figure 3.18: Failure of the ACM segmentation when the initial snake don't contain the object to segment (the astronaut face)

Topology changes: the ACM algorithm itself can not deal with more than one snake connected component, as you can see in figure 3.19. 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.

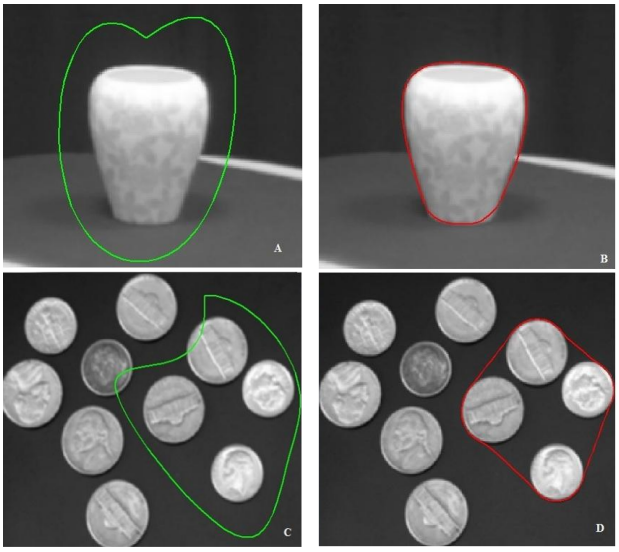


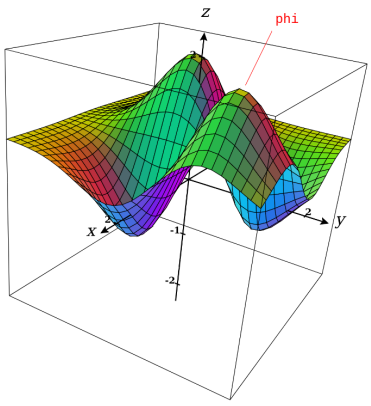
Figure 3.19: Failure of the ACM segmentation in topology changes. The green and red curves are the initial and final snakes, respectively

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

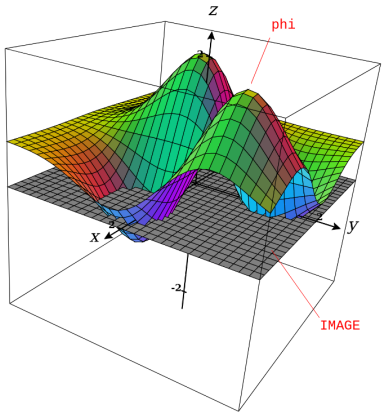
The ACM can accurately detect contours, but it is only really power and useful in a semi-automatic segmentation context or in a problem where objects don't vary enough in shape and position and a reasonable initial conditions cane be set. As long as a human manually draws as many initial contour as needed, 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

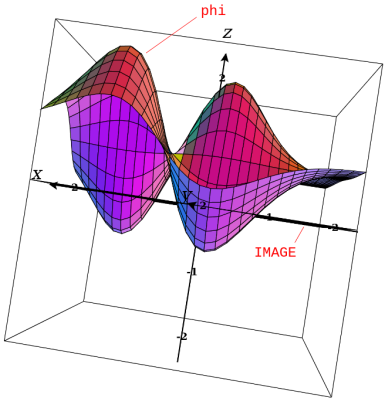
The algorithm proposed by Chan-Vese can be thought of as an "active surface model", as, instead of a curve, the idea is now to create a function $\phi(x, y)$, being (x, y) the image coordinates, and to evolve it over time. At each instant, the intersection between the surface $\phi(x, y)$ and the plane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



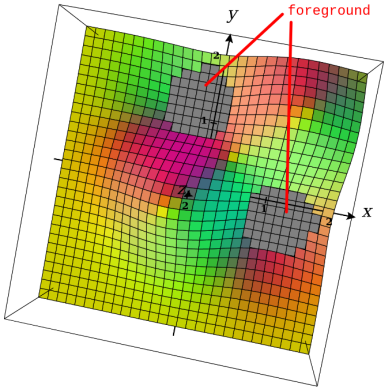
(a) The ϕ function



(b) The plane $z = 0$



(c) Intersection between ϕ and the plane $z = 0$



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

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.

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 3.21.

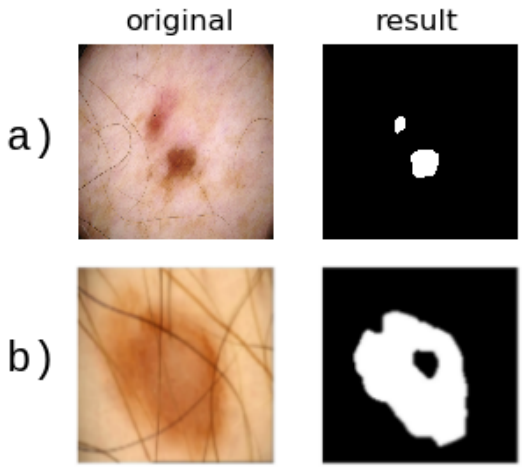


Figure 3.21: Success of Chan-Vese algorithm when segmenting: a) multiple connected components objects, b) objects with holes

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 [24].

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) need to be done in order to eliminate small false positive components.

Besides the figure 3.16, where the Chan-Vese segmentation performance on three hairs-removed images was showed, below you can take a look at an example of each specific step of the algorithm:

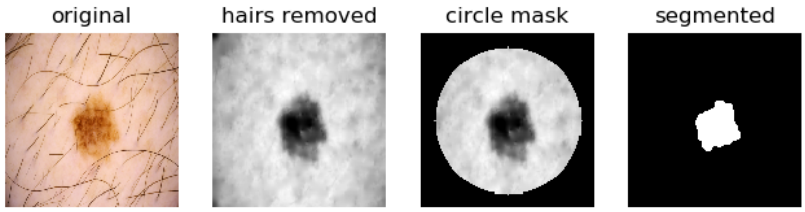


Figure 3.22: Chan-Vese segmentation steps

Once each image has been segmented and thus 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

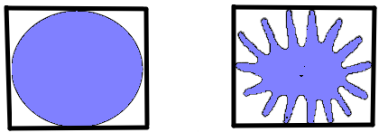
- Input: original and segmented images.
- Output: a 1D features vector for each image.

The Statistical Features Extraction (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.

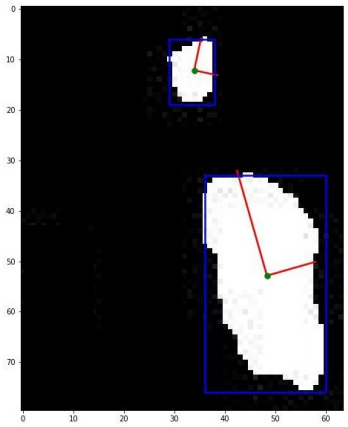
About the shape analysis, the idea is to extract, for each image, and for each connected component:

- one centroid,
- the eccentricity of the region,
- the major and minor axis of the ellipse that has the same normalized second central moment, and
- to quantify 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 **??**.



(a) The area ratio visualized



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

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 $c_0, c_1, ..., c_k$, 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 weighted mean for certain features and not for others. S can have many little connected components, that can deviate the *major* and *minor axis length mean* without being 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 handle 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 above described.

3.5 Topology

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 [32].

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 3.24: deforming a mug into a doughnut

But if they are so, than 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 (TANI). The toolkit has two main functionalities: extract a topological histogram and analyze in different sort of ways the textures, both applied on grayscaled images. The project lives in gitHub [17], 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 than 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 [16], different-dimensional elements can be easily handled as same-shaped arrays. Let's see how.

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

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

And J 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 not identify P anymore; they will instead refer to the r-dimensional entity ("*cell*" from now on) E with:

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

P happen 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 dimension 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 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 images processing and numerical features extraction 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, halfway, and output data we have worked on, and it is in this context that AI techniques helps.

In order to fulfill 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 combination. Likewise, four possible types of images input, (RGB, grayscale, HR, and segmented images) and so four different CNNs can be built. We will call all this possible networks, 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 reason because it may be sensible to do so:

- to satisfy the science hunger of knowledge: whichever could be the better model, study worse ones can lead 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, than that particular property influence to the system would be doubled, thus introducing a bias in the prediction, which can worsen it compared to the less inputs based MLP.

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 [3],

" 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 performance possible 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 phenomena that occurs when a network stops learning on inputs, to memorize them. 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 of 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 predictor capability is optimized for the training set only, thus taking steps back in generalization.

The processing 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 [3], 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 tool 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 get that ideal equilibrium point.

3.6.2 Architectures Design

Each submodel share with other of the same kind (MLP or CNN) inputs 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

MLP

For MLPs, a simple dense architectures have been used, with a Dropout and regularization layers to fight the overfitting, as explained above. The basic input and output layers structure, that will remain constant in each model is:

- a one-unit input layer: built upon each specific accepted input of the system. It guarantees the numerical variables normalization and the categorical values one-hot-encoding.
- a concatenation layer: that accepts and unify all normalized and encoded inputs.
- the output layer: 1-unit densely connected layer with sigmoid activation.

Below, I am going to describe the inner layers. The general models I have experimented with are:

- *model0*:
  - 16-units densely connected layer
  - 8-units densely connected layer with $l_1 = 0.001$, $l_2 = 0.001$
  - Dropout 0.3

CNN

For CNNs, a technique called *transfer learning has been used.*

Transfer learning

Transfer learning consists of applying in a considered problem a system that has already been trained in a different problem. Instead of designing the architecture and initialize every weight to a random number, transfer learning is the way of start from big networks that have been updated for long time.

Keras, the framework for machine learning I used to code all the AI functions of the project, provides a lot of pre-trained models in its applications module [14].

Same as before, only inner layer 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
  - 64-units densely connected layer with $l_1 = 0.001$, $l_2 = 0.001$
  - Dropout 0.5

3.6.3 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.

Stratified K-fold validation

For tuning the hyper-parameters of MLP and CNN architectures, a stratified 5-fold validation system has been designed to produce a plot of the means of *accuracy*, *AUC*, and *loss*, both for the train and valid 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? The dataset has to be split into training and validation sets. The training set tunes the network weights, while the validation set is for manually experimenting with different architectures and hyper-parameters.

The problem when splitting a dataset into training and validation dataset is that there is no knowledge about how the target and generally all values are distributed in them. Let's say we want the valid set to be the $x\%$, the 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 set into k folds, creates k models, and trains them upon K-1 folds and validates it upon 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.

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. It is still possible to get biased performances, but much it is less likely to happen.

Hyper-parameters Experimentation Workflow

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

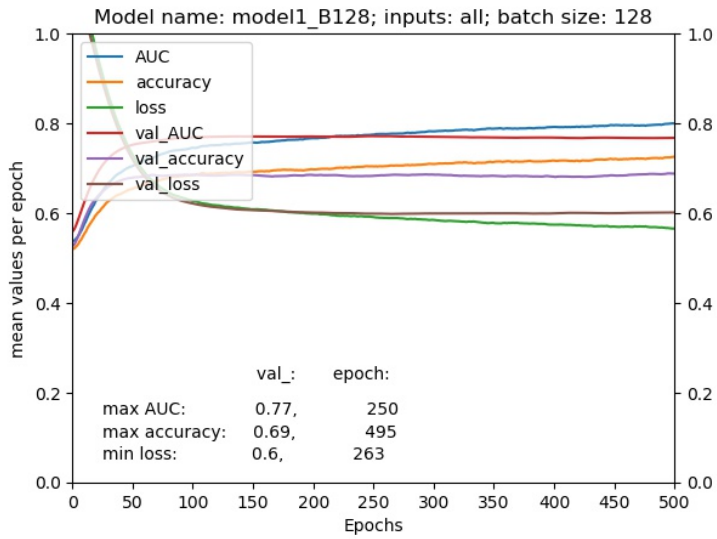


Figure 3.25: Experiment output plot sample

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 batches size from $2^0 = 1$ to $2^10 = 1024$ are proved and epochs manually adjusted to

make sure the extreme values are reached within the epochs time.

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

3.6.4 Evaluation metrics

The evaluation of an MLP is based on two values: the *area under the ROC curve* (AUC) and *accuracy*.

- the *accuracy* measures how often the algorithm classifies an input correctly. It seems simple, but the problem is that our MLP and CNN systems do not output a single integer number 1 or 0, bu 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 [18], 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):

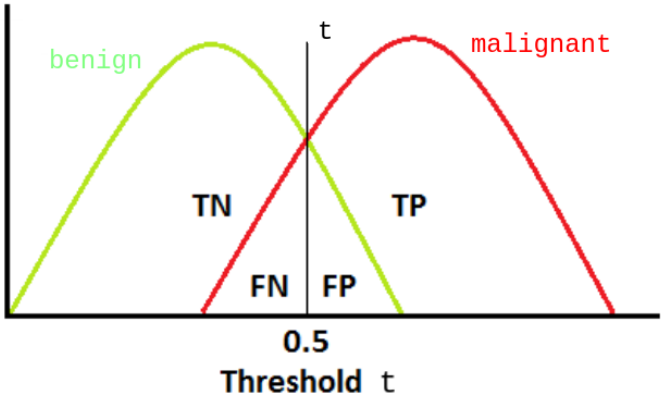


Figure 3.26: TPR and FPR depend on t

In other words, t variation displace those ratios, making the system tend 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 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:

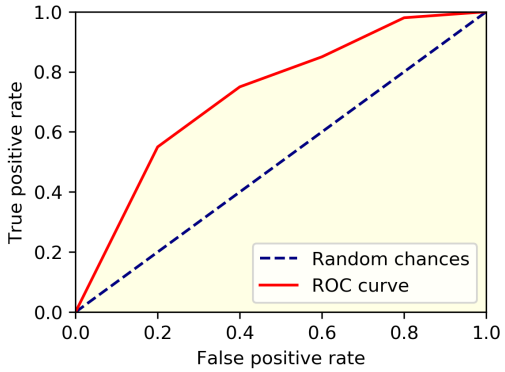


Figure 3.27: The ROC curve and the AUC, in light yellow

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 a good one should be near the ~ 0.8 value. If the system is performing a < 0.5 AUC than it is reciprocating the classes, misclassifying the cases.

Once stated and hopefully well described the methods adopted for the project, in the section 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 section **??**, we will talk about future possible works that may be done having this project as a start.

Chapter 4

Results & Discussion

Chapter 5

Appendices

5.1 Back-propagation algorithm

5.2 Convolution

Bibliography

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

[2] 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.

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

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

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

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

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

[8] 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/.

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

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

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

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

[13] 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.

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

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

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

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

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

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

[21] 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.

[22] 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.

[23] 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.

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

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

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

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

[28] André R.S. Marçal 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).

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

[30] Halil Murat Ünver and Enes Ayan. "Skin Lesion Segmentation in Dermoscopic Images with Combination of YOLO and GrabCut Algorithm". In: ().

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

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

[33] By Ihab Zaqout. "An Efficient Block-Based Algorithm for Hair Removal in Dermoscopic Images". In: (). 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.

5.3 Apuntes

- añadir al overview de las CNNs la información de las caapturas que está bastante bien:

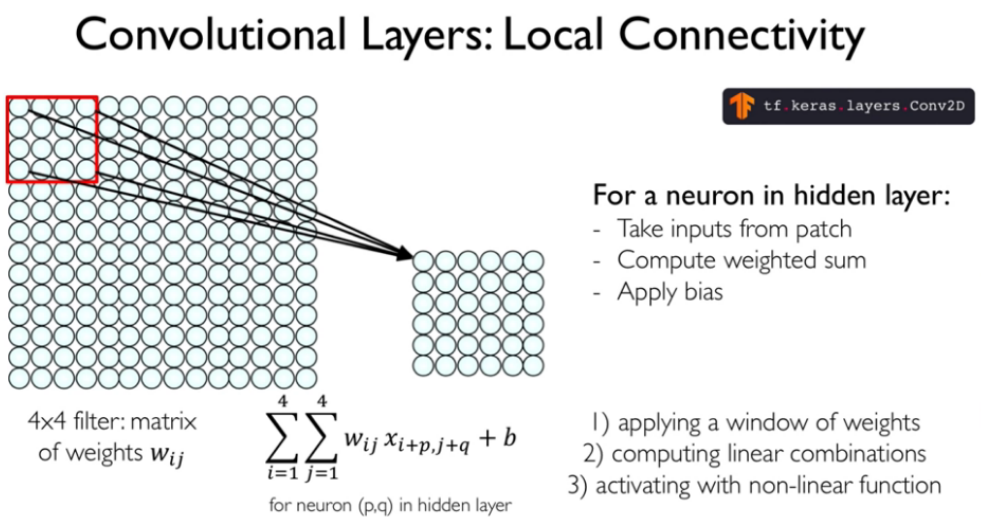


Figure 5.1: Diagrama de una CNN

- la línea no me convence, creo que produce algún error de rendering. Se supone que sirve para renderizar el símbolo >, pero es la primera vez que necesito un paquete para ello.

- Datos en España: En España, hay unos 4.000 casos de melanoma al año y más de 74.000 de cáncer cutáneo no melanoma. Cifras que se han duplicado en los últimos 30 años y que continarán haciéndolo si no se implementan unos hábitos de fotoprotección adecuados al tiempo que se revisa la piel.

- No hemos tratado mucho este aspecto: Using mages within the same patient and determine which are likely to represent a melanoma. Using patient-level contextual information may help the development of image analysis tools, which could better support clinical dermatologists.

- check the formal definition of convolution

- para convertir un ipynb a latex, una opción es:

  jupyter nbconvert /path/jupyterscript.ipynb –to latex

  Eso convertiría en automático las gráficas, pero el resto del documento no queda del todo bien renderizado. Por eso habrá que utilizar listings

y copiar y pegar el código si se desea que quede bien. Si no, se puede incluir un apéndice con todo el código y explicar en el interior del documento únicamente los resultados.