UNIVERSITÀ CATTOLICA DEL SACRO CUORE
INTERFACOLTÀ DI SCIENZE MATEMATICHE, FISICHE E NATURALI
E SCIENZE BANCARIE, FINANZIARIE E ASSICURATIVE

CORSO DI LAUREA IN APPLIED DATA SCIENCE FOR BANKING
AND FINANCE

TESI DI LAUREA MAGISTRALE

Image classification for Inspection Machines: a comprehensive review and application study

Relatore:
Dr. Enrico Barbierato
Corelatore:
Prof. Alfredo Marzocchi

Candidata:
Elisa Salvi

Matricola:
5005374

ANNO ACCADEMICO 2021/2022

To grandma Maria,
my guardian angel.

Contents

Introduction

The healthcare industry has made substantial progress during the past several decades embracing more innovations and cutting-edge technologies. The research of Computer Vision, imaging processing and pattern recognition enables more efficient diagnosis, treatment, health care operations, efficient medical records management and more, especially during emergencies. Machine Learning algorithms can handle complex data sets with applications in many fields like radiology, medical robotics, medical imaging, drug discovery, medical transcription and so on. The expectation for the near future is to see more and more healthcare companies and institutions experiment with technologies at the frontiers of knowledge to improve their services.

The progress of Computer Vision for healthcare applications is documented in a collection of articles called the "Journal of Healthcare Engineering" [1]. This journal enhances the development of these research fields and aims to bring together researchers, engineers, mathematicians and programmers in order to understand the problems usually encountered in these contexts through rapid online publications. This journal deals with several application engineering areas like cardiovascular, clinical, rehabilitation, neural, respiratory systems and so on. Moreover, the papers include topics such as biomedical imaging, image processing and sensors for bioinstrumentation.

Medical image technology is highly valuable to clinical analysis and treatment because it can provide important information about internal organs. Moreover, it can help doctors understand the patient's health condition and treat various diseases converting scanned images into interactive 3D models.

Computer Vision does not merely analyze medical images, but it also offers accurate data about phenomena that are hard to measure with traditional methods. A significant example is the case of the "Orlando Health Winnie Palmer Hospital for Women and Babies" [2]. One of the main causes of mortality in childbirth is postpartum hemorrhaging; this hospital is developing image analysis algorithms to precisely measure the blood loss during childbirth. In this way, the doctor estimation of the amount of blood lost during the delivery is improved in accuracy, minimizing human errors and improving treatment procedures for women.

Computer Vision is a branch of Artificial Intelligence (AI). The goal of Computer Vision is to develop digital systems to process, analyze and interpret visual data. Artificial Intelligence is a field of computer science that deals with brain-inspired software architectures for fast data-driven decision making and embraces Machine Learning, Computer Vision, natural language understanding, computational logic and processing. AI aims at building efficient algorithms for the recognition and the classification of objects, situations, and events. The challenge consists of under-

standing how the human brain processes information and apply the same methods to train specific software, making it capable of accurate and robust results. The incipit of "Artificial Intelligence" as a new discipline is defined in 1956, when a famous conference was held at Dartmouth College in Hanover, New Hampshire, America, and the organizers became the founders of AI research. This conference was proposed in August 1955 by J. McCarthy, M. L. Minsky, N. Rochester and C. E. Shannon. These researchers wrote a paper called "A Proposal for the Dartmouth Summer Research Project on Artificial Intelligence" [3] that suggested the examination of topics such as Neural Networks, computability theory, natural language processing and recognition. This was a highly stimulating meeting; relevant successes in the field of AI would be achieved by the very participants or their students. The participants were all prominent scientists of the time: the mathematician John McCarthy, the researcher in mathematics and neurology from Harvard Marvin Minsky, the director of information research at IBM Nathaniel Rochester, the mathematician and researcher in information theory Claude Shannon. The event was conducted as a brainstorming session where researchers could debate openly about the possibility to make machines behave like humans.

The scientific community agrees on the year 1956 as the first time the term "Artificial Intelligence" was coined, although substantial contributions to AI were made in previous years. AI had a strategic role during the Second World War. The idea that human intelligence could be simulated through the use of machines was born in the 1940s with the great contribution of the mathematician Alan Turing. The British scientist is considered one of the fathers of modern computing and his work had important applications especially in the field of cryptography. Turing collaborated with the British government to decipher the German military transmissions and develop the code behind the Enigma device. Moreover, he formulated the concepts of calculability, computability, and the Turing machine, that is a theoretical model of a machine capable of executing any kind of computable sequence. In 1950, Turing developed the "Turing Test", also known as the "Imitation Game", whose aim was to assess the presence of human intelligence in a machine. This work conducted by Turing, who died in 1954, played a key role in the Dartmouth conference.

The entrance of Computer Vision was marked by human visual system research conducted in various universities during the 1960s. The studies about how neurons react to different kind of stimuli yielded optimistic results. Scientists understood that human vision is characterized by a hierarchical analysis and detection of feature, like shapes, texture and edges. Furthermore, they took inspiration from how the human brain processes visual information to develop a Computer Vision system capable of achieving a complete understanding of an image. Many algorithms that are still relevant today were already described, such as edge detection, motion analysis and so on. The hype of these new projects was so high that researchers experienced difficulties to live up the public's expectations. The technical background failed to keep up with the complexity of the problems. This difficult period was called "AI winter" and scientist realized that their challenges would be solved at best in the next decades. The following years were characterized by more accurate mathematical researchers who laid the foundations of Computer Vision algorithms.

Computer Vision algorithms can handle both black and white and color images

and operate in many different areas; some examples of Computer Vision tasks are:

- image classification analyses an image and determines if it belongs to a certain class. This task often faces some challenges such as find the right classification for ambiguous images. This technique is used in the field of skin cancer detection to automate the identification process of cancerous tissue and reduce the chances of human error. The identification and the diagnosis of moles from melanomas are often difficult to the naked eye. Therefore, several algorithms are trained with a vast database of images consisting of both healthy and cancerous tissue.

- Object detection can identify particular entities within an image or a video. A concrete example is automatization of the damage detection on an assembly line, instead of training personnel for the inspection process that is way more expensive. Quality control is crucial to ensure the highest level of customer approval and limit possible complications after the sale of the product. In the medical field, this method is applied in x-ray radiology, ultrasound, endoscopy and many more procedures in order to help medical professionals identifying any issues or abnormalities in patients' internal organs.

- Image segmentation divides the image into sections and analyzes the delimited areas individually. This method is useful to highlight the pixels of a tumorous section of a tissue in a medical report. Tumors spread quickly to other parts of the body, so making early detection is especially crucial. It can reduce the possibility of human error and save a patient's life by identifying even the slightest difference with a high level of certainty.

- Action and emotion recognition concerns the identification of specific actions and sentiments showed in an image or a video. This technique can be used to assess the quality of a surgery by measuring the level of activity, analyzing and detecting medical professional movements.

- Object tracking detects and monitors an object in real-time video feeds. This technique is used, for example, for autonomous vehicles to detect, classify and track other cars in motion to avoid collisions. This is also crucial for to develop monitoring systems in order to identify any risky situations for the workers or the environment.

- Image editing modifies an image, for example to obscure sensitive data for privacy issues.

- Content-based image retrieval searches images from large data collections, relying only on the content of the images.

The Covid pandemic presented a tough challenge for the global healthcare system and has required worldwide researchers to elaborate and implement new strategies. Computer Vision has provided a significant contribution against this biological threat [4]. The previous listed tasks can be applied to the diagnosis, control, treatment, and prevention of Covid-19 infections. Several tools are implemented to

monitor and prevent the spread of the pandemic, for example masked face detection software, radiography images analysis, remote patients monitoring and occupancy detection to ensure social distance. These techniques are adopted more and more by healthcare facilities to improve service quality and increase efficiency. In January 2021 the first open-source network, called "COVID-Net", was designed to handle Covid-19 hospitalization data, like age, sex, ethnicity and underlying health conditions of Covid patients, and also allowing to weekly update hospitalization rates.

However, the implementation of Computer Vision algorithms deals with some critical issues. Firstly, in real life the datasets are not sufficiently large to train the algorithms properly. Consequently, "Data Augmentation" technique, explained in chapter 1.1, are often used in order to develop algorithms robust to image transformations, like non-optimal brightness conditions, partial coverage of the subject, scale variations, blurring and so on. Moreover, the process of image feature extraction can be negatively influenced by inconspicuous features.

Despite these difficulties machine vision systems have numerous applications. In the industrial and manufacturing sectors, Computer Vision has an important role thanks to the possibility of being integrated directly on production lines and in factory environments. This technology is exploited for many areas of 4.0 industry, like monitoring and predicting the maintenance of industrial assets or classifying anomalies on the surface of mechanical component [5].

This dissertation reviews the most used techniques for image preprocessing, feature extraction and feature engineering and aims to build, analyze and compare several models for the classification of a database composed of visual imagery.

The preprocessing techniques are presented in chapter 1 and are fundamental to improve the quality of input images for the analysis success. These methods include Data Augmentation (see section 1.1), different type of filters for blurring and smoothing the images (see section 1.2), Edge Detection, to identify automatically objects boundaries in an image (see section 1.3), and Hough Transform, to find a particular shape within an image (see section 1.4).

Chapter 2 focuses on widely used feature extraction algorithms developed specifically for image processing, such as RGB color levels extraction (see section 2.1), Histogram of Oriented Gradients, or HOG, that describes the structure of an object with the distribution of intensity gradients and the edge directions (see section 2.2), Scale Invariant Feature Transform, or SIFT, that is used to detect and describe local features that are invariant against various transformations (see section 2.3), and Haralick Texture, to analize the surface or the substance of the represented object (see section 2.4).

In chapter 3 is explained how features are optimized, after the extraction, in order to find the best feature to successfully perform a Machine Learning algorithm. Different types of feature selection techniques are introduced, especially Filter methods (see subsection 3.1.1), Wrapper methods (see subsection 3.1.2) and Embedded methods (see subsection 3.1.3). Moreover, features can be manipulated in order to create new features that improve the quality of a Machine Learning model. This field is called "Feature Engineering" and includes both Manual and Automated algorithms (see chapters 3.2.1 and 3.2.2).

The core of this dissertation is the classification techniques for the categorization

of images into a number of classes. Computer Vision is based on image classification task, such as for object detection, and the algorithms that solves this kind of issue are called "classifiers". In chapter 4 are presented different widely used classifiers, such as Decision Tree classifier (see section 4.1), Random Forest (see section 4.2), K-Nearest Neighbors, or KNN (see section 4.3), Support Vector Machine, or SVM (see section 4.4), Multiple Classifier (see section 4.5), Convolutional Neural Networks, or CNNs (see section 4.6), and Transformers (see section 4.7). Is important to analyze various algorithms because it is impossible to determine the best way to classify an image a priori, this depends on various features that characterize the data.

Finally, in chapter 5 an application example about the classification for closure caps of pharmaceutical ampoule is described as a practical demonstration of the explained theoretical background presented in the previous chapters.

Chapter 1

Image Preprocessing

The word "image" comes from the Latin idiom "imago" and refers to a two-dimensional picture that represents a physical object produced in electronic form. In computer science an image is a binary matrix representation of visual information that can be process and visualized on a screen. Digital images are composed of pixels [6]. "Pixel" means "picture element" and is the basic, logic and square-shaped unit of an image represented on a digital display device. In other words, pixels are combined using geometric coordinates to form visual imagery on a computer screen. The resolution of an image depends on the size of the pixel chosen, and consequently on the size of the grid used to subdivide the image that is to be represented. In a low-quality image, the pixel number is small, but their size is large, so the picture will look blocky or pixelated.

In black and white images, pixels are binary so each pixel can be one, to represent black, or zero for white. In colored images, four colors are mixed together to form every possible color. These basic colors are white, blue, green and red, and they are represented in binary notation. The number of bits used to store each pixel is called "color depth". If an image has a large color depth, more colors will be available and its representation will be more accurate and looks better when zoomed, but the image will be stored in a larger file occupying more memory. PC and smartphone monitors use 24-bit color system in order to ensure good quality even if images are zoomed. Moreover, human eyes are limited to about ten million colors, so more bits than 24 are not necessary. The most diffused color system in Computer Vision programming is in 8-bit, so the palette includes 256 colors.

"Data preprocessing" is an essential concept useful to achieve accurate results in the field of Machine Learning (ML). The aim of ML is to use the input data to train an algorithm and provide the best results. The whole process is based on the input data, so the most important thing is to validate data to feed in at the beginning to avoid the well-known problem of "garbage in, garbage out". Data preprocessing is the concept of cleaning the initial dataset checking for instance missing values and possible noises, eliminating outliers, normalizing units of measure and correcting errors.

The aim of image preprocessing is to improve the quality of input images for the analysis success. This includes resizing, color corrections, filtering, blurring, augmentation and so on. Image preprocessing may also speed up the training process

of the model. For some ML algorithm, preprocessing is mandatory: for example, Convolutional Neural Networks with fully connected layers require the same size for all the input images.

1.1 Image Augmentation

Deep learning algorithms perform well when the amount of data is large enough [7]. Actually, is not always possible to have lot of data to feed in training algorithm for most of the analysis. The lack of data may lead to an overfitting of the training process and the manually collection of data is costly and not always possible. The word "augmentation" literally means "the action or process of making or becoming greater in size or amount" [8].

Image augmentation is a technique that changes the initial data to create more input data to feed the model. In other words, image augmentation artificially expands the available dataset generating new images from existing ones. This is obtained by using techniques such as random rotation, shifts, scaling, cropping, flips and so on [9]. In image classification, augmentation is crucial to build a strong classifier and also to reduce the skewness of data available in the various classes. Moreover, image augmentation is usually required to boost the performance of Artificial Neural Networks. Image augmentation can be divided in two main categories: spatial augmentation and pixel augmentation. The most used techniques [9] for image spatial augmentation are:

- Rotation: is the most common technique for image augmentation because the images information remains the same at each different angle. The calculations of the new coordinate of each pixel in the input image can be done independently. The coordinates of a point (x_1, y_1) when rotated by an angle θ around (x_0, y_0) become (x_2, y_2) [10], as shown in 1.1:

$$\begin{cases} x_2 = \cos\theta(x_1 - x_0) + \sin\theta(y_1 - y_0) \\ y_2 = -\sin\theta(x_1 - x_0) + \cos\theta(y_1 - y_0). \end{cases} \tag{1.1}$$

  These equations can be rewritten in a compact way, as shown in 1.2:

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x_1 - x_0 \\ y_1 - y_0 \end{bmatrix}. \tag{1.2}$$

- Translation: image shifting left, right, up, or down is a geometric transformation that reallocates every object of an image after mapping their positions. So, the position of the objects in the image change and the use of this technique can result in a more generalized model. This technique can be a very useful transformation to avoid positional bias in the data [9]. A pixel in an image located at (x_1, y_1) is shifted to a new position (x_2, y_2), as shown in 1.3,

by displacing it through the translation (β_x, β_y), that is randomic.

$$\begin{cases} x_2 = x_1 + \beta_x \\ y_2 = y_1 + \beta_y \end{cases} \tag{1.3}$$

- Flipping: in both left-right direction and up-down direction. This can't be used for images containing some form of alphanumeric text because it will create an unreal case. The reflaction of a pixel in an image can be made multiplying a reflection matrix R by the initial coordinates, as shown in 1.4:

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = R \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}. \tag{1.4}$$

  The reflection matrixes R in the x-axis, y-axis, in the origin and in the line $y = x$ are respectively:

$$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}.$$

- Cropping: randomly cropping a central patch of each image provides an effect very similar to translations, but cropping reduces the size of the input. Cropping images can be used as a practical processing step for image data with mixed height and width dimensions [9].

- Noise injection: is the creation of new images adding noises, usually from Gaussian distribution, to the initial image in order to build a noise-robust classifier [9].

These geometric transformations are easy to implement and are able to solve the problem of positional biases in the training data. For example positional biases are present in a facial recognition dataset, where every face is perfectly centered in the frame. The disadvantages of geometric transformations are additional memory required, transformation compute costs, and additional training time.

Another technique used for image augmentation is "pixel augmentation" that involves brightness, contrast, saturation and hue random changes in order to avoid lighting biases. An example of pixel transformation consists of restricting pixel values to a certain minimum or maximum value. The disadvantages of color space transformations are the same cited before for geometrical transformations. Additionally, color transformations may discard important color information, for example, after a pixel values change, objects in the image may become impossible to recognize.

To sum up, data augmentation manipulates images and creates different versions of similar content in order to expose the model to a wider array of training examples. Note that image augmentation is used only for the training process, while other preprocessing techniques, like filtering, are applied both at training and testing set.

Data augmentation is an important research topic and is still under development. It can concern important task such as cancer type classification that is characterized

by a serious lack of data. Jason Wang and Luis Perez, two researchers from Stanford University, have implemented a method called "neural network augmentation" [7] showing how a neural network can learn augmentation that best improves the classifier minimizing loss. "Augmenting data via a neural net is achieved by concatenating two images of the same class to create an input of 6 channels deep (2 if gray scale)", explain the two researchers in their paper. Thanks to this method, a dataset of size N can create N^2 pairs, which results into a magnitude increase. Moreover, this type of data augmentation can be combined to others data augmentation techniques.

1.2 Image Filtering for Blurring and Smoothing

By "image filtering" is meant the changing in appearance of an image by modifying the pixels colors [11]. In other words, filtering changes the pixel values of an image: meaning that each pixel is individually affected and the colors are altered without changing the pixel positions. It can be performed on both 2-D and 3-D images. The main targets of image filtering are noise reduction and resolution enhancement using contrast changes, brightness adjustment, blurring, sharpening, smoothing and edges empathizing. Spatial filters can be divided into two main categories:

- Linear filters: that blur image details and edges. An example can be Gaussian filter, Laplacian filter and Averaging filter. In this case the value of an output pixel is a linear combination of the values of the pixels in the input pixel's neighborhood.
- Non linear filters: are used to detect edges. An example of edge-preserving filter is the Median filter.

1.2.1 2D convolution

Images can be filtered for noise removing and image blurring. The mathematics behind filters can be expressed using 2D convolution. The 2D convolution is an operation that starts with a kernel, which is simply a small matrix. This kernel "slides" over the 2D input data, the matrix of the image's pixels, performing an elementwise multiplication with the part of the input it is currently on, and then summing up the results into a single output pixel. The kernel repeats this process for every location it slides over, converting a 2D matrix of features into yet another 2D matrix of features. The output features are essentially, the weighted sums (with the weights being the values of the kernel itself) of the input features located roughly in the same location of the output pixel on the input layer. This means the size of the kernel directly determines how many input features get combined in the production of a new output feature.

Image blurring is achieved by convolving the image with a filter kernel. It is useful for removing high frequency content, like noise and edges, from the image.

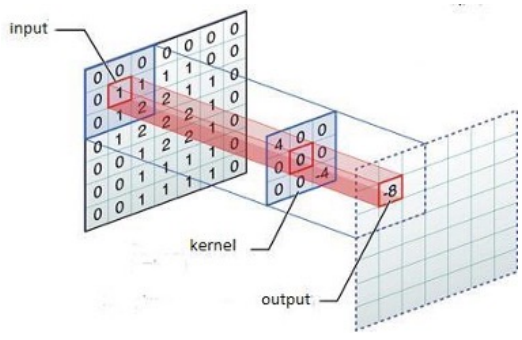


Figure 1.1: 2D convolution.

So, edges are blurred a little bit in this operation (there are also blurring techniques which don't blur the edges).

Linear filter is the most used type of convolution, where an output pixel's value is a weighted sum of pixel values within a small neighborhood N. The resulting matrix can be expressed as:

$$g(i,j) = \sum_{k,l} f(i+k, j+l)h(k,l) = \sum_{k,l} f(i-k, j-l)h(k,l) = \sum_{k,l} f(k,l)h(i-k, j-l) \tag{1.5}$$

or in a continuous version, as shown in 1.6:

$$g(x) = \int f(x-u)h(u)\,du \tag{1.6}$$

where $h(k,l)$ is the kernel, or mask, composed of the filter coefficients and $f(i+k, j+l)$ is the input matrix. The formula 1.5 can be interpreted as the addition of shifted functions $h(i-k, j-l)$ multiplied by the input pixel values $f(k,l)$. This can be expressed with the convolution operator as shown in 1.7:

$$g = f \otimes h. \tag{1.7}$$

Convolution has both commutative and associative properties and is linear shift-invariant operator. Note that convolution produces a resulting matrix that is smaller than the original image, to deal with this, a number of different "padding" or extension modes have been developed:

- zero: set all pixels outside the source image to 0;
- constant: set all pixels outside the source image to a specified border value;
- mirror: reflect pixels across the image edge;
- replicate (or clamp): repeat edge pixels indefinitely;
- extend: extend the signal by subtracting the mirrored version of the signal from the edge pixel value;

and so on. The simplest filter to implement is the moving average filter, which simply averages the pixel values in a K × K window. This is equivalent to convolving the image with a kernel of all ones and then scaling. To sum up, in linear filters each output pixel is a weighted summation of some number of input pixels. This is the easiest approach, but in many cases, a better performance can be obtained by using a non-linear combination of neighboring pixels. For example non-linear filters are useful when the image has occasionally very large values of noise. In this case, regular blurring with a Gaussian filter fails to remove the noisy pixels turning them into softer, but still visible, spots. Next subsections will describe on both linear filters, as averaging and Gaussian filter, and non-linear filters, as median and bilateral filtering. These topics are examined more in depth in chapter three of the book "Computer Vision: Algorithms and Applications " [11].

1.2.2 Averaging

The simplest linear filter to implement is the average filter, which simply averages the pixel values in a $k \times k$ window. This is equivalent to convolving the image with a kernel of all ones and then scaling. In other words, averaging method is done by convolving an image with a normalized box filter. It takes the average of all the pixels under the kernel area and replaces the central element. We should specify the width and height of the kernel. The function smooths the image using the kernel:

$$K = \frac{1}{ksize.width * ksize.height} \begin{bmatrix} 1 & 1 & 1 & ... & 1 & 1 \\ 1 & 1 & 1 & ... & 1 & 1 \\ ... & & & & & \\ 1 & 1 & 1 & ... & 1 & 1 \end{bmatrix}.$$

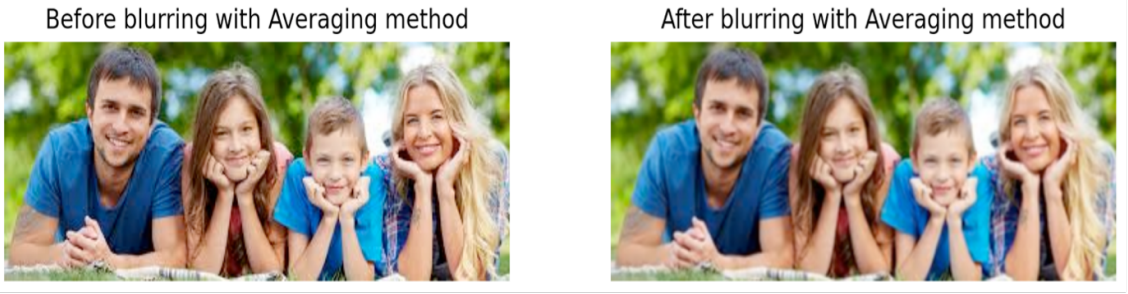


Figure 1.2: Example of averaging filtering.

1.2.3 Gaussian Blurring

Another widely used linear filter is the Gaussian filter. In image processing, a Gaussian blur is the result of blurring an image by a Gaussian function. In two dimensions, a Gaussian function is:

$$G(x, y) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{1.8}$$

where x is the distance from the origin in the horizontal axis, y is the distance from the origin in the vertical axis, and sigma σ is the standard deviation of the Gaussian distribution. Values from the distribution 1.8 are used to build a convolution matrix which is applied to the original image. Each pixel's new value is set to a weighted average of that pixel's neighborhood, as shown in 1.9:

$$GB[I]_p = \sum_{q \in S} G_\sigma(||p - q||) I_q \tag{1.9}$$

where $G_\sigma(||p - q||)$ is the normalized Gausian function. $GB[I]_p$ is the result at pixel p and the right hand side is essentially a sum over all pixels q weighted by the Gaussian function. I_q is the intensity at pixel q. In short, this method takes the neighborhood around the pixel and finds its Gaussian weighted average using

Figure 1.3: Example of Gaussian blurring.

a Gaussian kernel. This filter is a function of space: nearby pixels are considered while filtering. It doesn't consider whether pixels have almost the same intensity and whether a pixel is an edge pixel or not. So, it blurs the edges also. In the filtered image noise and detail are reduced.

1.2.4 Median Blurring

Median filtering is a non-linear filter that takes the median of all the pixels under the kernel area and replaces the central element with the median value. This is highly effective against "salt-and-pepper" noise, called impulsive noise. Moreover, median filter is able to preserve edges for moderate level of noise and this is of critical importance for the visual appearance of the image. Because of this, it is widely used in digital image processing.

Figure 1.4: Example of median filtering.

1.2.5 Bilateral Filtering

A bilateral filter is a non-linear, edge-preserving, and noise-reducing smoothing filter for images. This method can reduce unwanted noise very well while keeping edges fairly sharp. It has been used in various contexts such as denoising texture editing and relighting, tone management, stylization and optical-flow estimation. However, it is very slow compared to other filters. Its formulation is simple: each pixel is replaced by a weighted average of its neighbors, as shown in 1.10. It depends only on two parameters that indicate the size (sigmaSpace) and contrast (sigmaColor) of the features to preserve. So, it preserves the edges since pixels at edges will have large intensity variation.

$$BF[I]_p = \frac{1}{W_p} \sum_{q \in S} G_{\sigma_s}(||p-q||) G_{\sigma_r}(||I_p - I_q||) I_q \tag{1.10}$$

In the formula 1.10, $\frac{1}{W_p}$ is the normalization factor, $G_{\sigma_s}(||p-q||)$ is the space weight and $G_{\sigma_r}(||I_p - I_q||)$ is the range weight. The normalization factor and the range weight are new terms added to the Gaussian blur equation. σ_s denotes the spatial extent of the kernel (the size of the neighborhood) and σ_r denotes the minimum amplitude of an edge. It ensures that only those pixels with intensity values similar to that of the central pixel are considered for blurring, while sharp intensity changes are maintained. The smaller the value of σ_r, the sharper the edge. As σ_r tends to infinity, the equation tends to a Gaussian blur.

Figure 1.5: Example of bilateral filtering.

1.3 Edge Detection

In the fields of image preprocessing and Computer Vision a fundamental technique for feature extraction and image segmentation is edge detection. The points where the image brightness varies sharply are called the edges (or boundaries) of the image. Edge detection is used to identify automatically objects boundaries in an image by detecting discontinuities in brightness and sudden changes in neighboring pixel intensity. Discontinuities are places in an image where brightness differs suddenly and are enhanced by mathematical models. Edges are an important feature that helps in the interpretation of the image structure. The main two methods for edge detection are Sobel and Canny edge detection.

Sobel edge detection is based on the Sobel operator that was developed in 1968 by Sobel and Feldman [12], two scientists and researchers in digital image processing from Stanford University. This operator emphasizes regions characterized by sudden changes in pixel intensity that correspond to edges performing a 2-D spatial gradient measurement on an image. Edges can be detected computing the first derivative of the intensity function and looking at where the gradient is higher than a particular threshold value. That's because a sudden change in the derivative leads to a change in the pixel intensity. The Sobel operator consists of two 3×3 convolution kernels where the second has the same values of the first one, but is rotated by 90 degree:

$$\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \textit{and} \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & 2 & -1 \end{bmatrix}. \tag{1.11}$$

The kernels 1.11 produce separately two measurements of the gradient components in x and y orientation, respectively G_x shown in 1.12 and G_y shown in 1.13.

$$G_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \otimes I \tag{1.12}$$

$$G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & 2 & -1 \end{bmatrix} \otimes I \tag{1.13}$$

In 1.12 and 1.13, I is the input image and $\otimes$ is the convolutional product. G_x and G_y can be also written as shown in 1.14 and 1.15.

$$G_x = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 & -1 \end{bmatrix} \otimes I \tag{1.14}$$

$$G_y = \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 2 & 1 \end{bmatrix} \otimes I \tag{1.15}$$

The absolute magnitude of the gradient is found combining the two values G_x and G_y and is given by:

$$|G| = \sqrt{G_x^2 + G_y^2}. \tag{1.16}$$

The formula 1.16 is often approximated in order to be computationally faster as:

$$|G| = |G_x| + |G_y|. \tag{1.17}$$

Moreover, the angle of orientation of the gradient can then be approximated as:

$$\theta = \arctan \frac{G_y}{G_x}. \tag{1.18}$$

Performing only the vertical kernel, the resulting image will have edges enhanced in the x direction. Vice versa, performing a horizontal kernel, the resulting image will have edges enhanced in the y direction. In other words, the Sobel image in the x direction predominantly identifies vertical edges, so the areas whose gradient is largest in the horizontal direction, and vice versa. In the Sobel image, where the gradient is performed in both directions, the structural integrity remains intact, and the original image is transformed into an edge structure representation.

A more robust and flexible algorithm for edge extraction is Canny edge detection developed by John F. Canny [13], Ph.D. in electrical engineering. This is a multi-step process composed of four steps for boundaries extraction, which consist of:

- noise reduction: applying a Gaussian blurring filter;
- Sobel kernel: to compute the intensity gradient of the image;
- non-maxima suppression: to keep only the local maxima of gradient magnitude pixels pointing in the gradient direction;
- hysteresis thresholding: defining and applying the upper and lower thresholds T_u and T_l.

Firstly, is important to minimize noises using a Gaussian blurring filter before performing the Canny edge detection to remove unnecessary details that could distract from the actual boundaries. Secondly, the intensity gradient of the smoothed image is calculated by filtering it with a Sobel kernel, both horizontally G_x and vertically G_y, and the intensity gradient magnitude G and the direction θ are computed for each pixel, as shown in 1.17 and 1.18. The Sobel method may be susceptible to noise, so other steps are implemented to extract better edges.

Now a technique called "non-maximum suppression of edges" is used to fully scan the image in order to filter out unwanted pixels which may not constitute a boundary. This is an edge thinning process made by comparing each pixel with its 3×3 neighboring pixels in the gradient direction. The gradient magnitude of the selected pixel is preserved when is greater than its neighboring pixels, otherwise is set to zero. In other words, is checked if every pixel is a local maximum in its neighborhood.

The last step is called "Hysteresis Thresholding" and involves the definition of a lower and an upper threshold T_u and T_l to ignore regions that are not edges. So, each pixel gradient magnitude is compared with these two thresholds. When the gradient magnitude is higher than the upper threshold, so $G > T_u$, the pixels are surely edges and are included in the final edge map as "strong edges". Vice versa,

when the gradient magnitude is smaller than the lower threshold, so $G < T_l$, the region is immediately discarded, and the pixels are excluded from the final edge map. All the pixels with a gradient magnitude that falls into the two thresholds range, are called "weak edges" and need an additional test. If a weak pixel is connected to a strong edge, then is marked as an actual edge and is included in the final edge map, otherwise is discard.

In the Canny function, the two thresholds can be manually set, but this is not trivial. With a too tight range, many actual edges will be discarded, and the structure of the image can result incomplete. Canny edge detection is the most convenient and widely used method for performing edge detection because has the advantage of combining different technique to come up with a robust result.

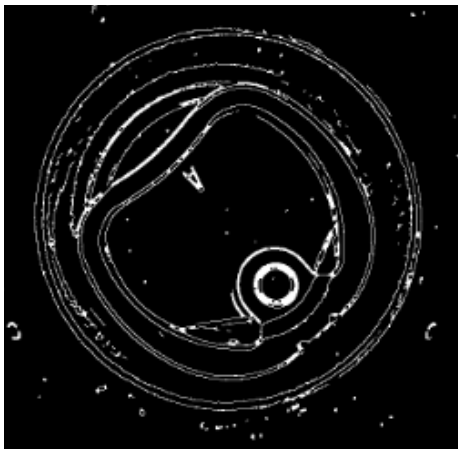


Figure 1.6: Example of Canny Edge Detection performed on a pharmaceutical ampoule cap.

1.4 Hough Transform

The main problem of boundary detection is knowing which edges in an image correspond to the actual boundary. The Hough transform is an extraction technique used in the field of digital image processing and designed in 1972 by R. Duda, professor of electrical engineering at San Jose State University, and P. E. Hart, a computer scientist [14]. This technique is used to isolate features of a particular shape within an image and is able to describe boundaries using a small number of parameters. In its classical form it is based on the recognition of lines in an image, but has been extended by including the recognition of other arbitrarily defined shapes, like lines, circles, ellipses, and so on. This method became famous in 1981 with "Generalizing the Hough transform to detect arbitrary shapes" [15], a journal article by D. H. Ballard, professor of computer sciences at the University of Texas, Austin. This technique aims to find the shape class of objects in an image even if in boundaries are present some imperfections or gaps. This is made performing a voting procedure.

When a Canny edge detection is performed on an image, the resulting image may have extraneous data, noises, and incomplete edges. Hough transform deals with these problems using more than an algorithmic approach. Firstly, consider the straight-line detection problem: the idea is to find lines from a cluster of points and assume that the equation of the straight line is:

$$y = mx + c \tag{1.19}$$

where m is the slope and c is the intercept. Consider now one point (x_i, y_i) exactly on the straight line 1.19. Now y_i can be written as:

$$y_i = mx_i + c. \tag{1.20}$$

The equation 1.20 can be rewritten as:

$$c = -mx_i + y_i \tag{1.21}$$

where x_i and y_i are known, so this is a straight-line equation in m and c. The equations 1.20 and 1.21 allow to look at the problem in two spaces: the image space (the x, y space) and the parameter space (the m, c space). In this way, taking a particular point (x_i, y_i) that lies on a line in the image space and plug this point into the parameter space equation, may result in a line in the parameter space. All the points that lie on this resulting line are (m, c) values that correspond to all the lines in the image space that pass through the point (x_i, y_i). Another point (x_j, y_j) in image space, gives another line in parameter space that corresponds to all the straight lines that pass through (x_j, y_j). The line of interest in the image space is the one that pass through both (x_i, y_i) and (x_j, y_j): this is the line described by (m, c), the point of intersection between the two lines in parameter space. Considering more points on the image space line, the resulting lines in parameter space passes through (m, c). Vice versa, taking a point that does not lie on the image space line of interest, it would create a line in parameter space, but that line is not going to pass through (m, c). In other words, a point in image space maps a line in parameter

space and a line in image space ends up being a point in parameter space, and vice versa. This is the main idea of the line detection algorithm. The first step is to quantize the parameter space (m, c) creating an accumulator array $A(m, c)$, that is a discrete space. Initially, all the array is set to 0, so:

$$A(m, c) = 0 \; for \; all \; (m, c). \tag{1.22}$$

Then a point on an edge (x_i, y_i) is considered and plugged into the equation of the straight-line 1.21. For every (m, c) that fall on this straight line we are going to increment the accumulator array:

$$A(m, c) = A(m, c) + 1. \tag{1.23}$$

This process is repeated for another point to find the point of intersection that will have a value of two in the accumulator array. Essentially the points in the image space are mapped into lines in the (m, c) space and then there is a voting along these lines, called a voting scheme. The (m, c) point that has more votes, so the local maxima of $A(m, c)$, is the point of intersection, that correspond to the line of interest in the image.

There is a problem considering the equation 1.19: $m \in [-\infty, \infty]$ and so span all possible values of m is not computationally possible. In these cases is preferable to use for the image space the line parametrisation equation:

$$x \sin\theta - y \cos\theta + \rho = 0. \tag{1.24}$$

In equation 1.24, θ is the orientation that is finite ($0 \leq \theta < \pi$) and ρ is the distance of the straight line from the origin that is finite and can't be bigger than the size of the image itself. Now the parameter space is the (ρ, θ) space and a point in the image space will be a sinusoid in the parameter space. The intersection of the sinusoids describes the parameters of the straight line of interest. In this case, an accumulator array is created with the parameters (ρ, θ) and the algorithm votes along sinusoids in that space. An issue of this method may be the choice of the accumulator array size: a too big accumulator array may marge different lines. On the other hand, a too small accumulator array may fail the vote and many lines would be missed because of noises. And if it's too small, then in the presence of noise and quantization and other effects in the image side, you may not have any cell that gets high enough number of votes because of getting scattered around this very high resolution of cells in the accumulator array. Another issue is how many lines actually has the image: the accumulator array may present lots of high values close to each other, so a peak finding algorithm is useful to count the peaks after the voting process. Moreover, vote for a batch of cells rather than a single point makes the algorithm more resilient to noise in a real implementation.

The Hough transform can detect also circles. The "circle Hough transform" is a basic feature extraction technique used for detecting circles in imperfect images. The circle candidates are produced by voting in the Hough parameter space and then selecting local maxima in an accumulator matrix. In more detail, let's consider a set of points that lies on a circle:

$$(x_i - a)^2 + (y_i - b)^2 = r^2 \tag{1.25}$$

where (a, b) correspond to the center of the circle and r to the radius. These are the three parameters that are interesting given a set of edges.

If the radius is known, the 2- dimensional accumulator space will be $A(a, b)$. So, given any (x_i, y_i) and that r is known, in the parameter space the equation 1.25 will be a circle in (a, b):

$$(a - x_i)^2 + (b - y_i)^2 = r^2. \tag{1.26}$$

In other words, a point in the image space results in a circle in the parameter space and the accumulator array $A(a, b)$ votes along a circle for any given point in image space. Taking other image points leads to circles with an intersection at (a, b) point corresponds to the (a, b) values of the circle that passes through all the points in the image space.

If not only the radius r and the edge locations (x_i, y_i) are known, but also the edge directions ψ_i is known, then the center of the circle lies on the edge direction. With this information, the vote in the accumulator array will be no longer on the entire circle, but only on two points which correspond to the two points at distance r in both direction of the edge in the parameter space. Two votes are needed because the direction on which the center lies is unknown and these can be written as:

$$a = x_i \pm r \cos \psi_i \tag{1.27}$$

$$b = x_i \pm r \sin \psi_i. \tag{1.28}$$

So only the incrementation of two points in the accumulator array is needed for each point in the image space. Repeating this process for all the points of the image space, the point that receives the maximum number of votes is the center of the circle.

In the case where the radius r and the edge directions are not known, the accumulator array will be $A(a, b, r)$, so 3-dimensional. So, for any given (x_i, y_i) the equation 1.26 corresponds to a cone in the space of parameter (a, b, r) and for each point in the image space the vote will be along a cone. Note that if the number of parameters goes up in accumulator array space or parameter space, the amount of work involved increases exponentially.

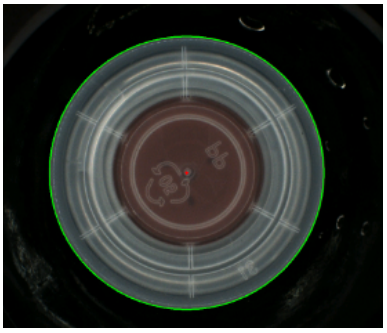


Figure 1.7: Example of Circle Hough Transform performed on a pharmaceutical ampoule cap.

Chapter 2

Feature Extraction

Literally, the word "feature" means a "distinctive attribute or aspect of something" and is something to pay special attention. In other words, a feature is a unique quality or characteristic that clearly helps to identify a particular object. Feature extraction is the process of transforming raw data into more manageable information describing the original dataset in an accurate and complete way. This method selects, combines and creates new variables from the existing ones while preserving the main information of the original data set.

The aim is to achieve better results than performing a Machine Learning algorithm directly to the raw data. Feature extraction is also useful for reducing the amount of redundant data summarizing most of the information in order to speed up an analysis. Feature extraction is a high interest topic and a significant number of dedicated methods are developed. Moreover, feature extraction is fundamental to achieve a good performance in different types of analysis, like classification, clustering, recognition, and detection.

Feature extraction can be implemented both manually and automatically. When is the data scientist who identifies and extracts the relevant features by implementing an algorithm, we talk about manual feature extraction. Lately this method has been replaced by deep networks. This technique extracts features automatically from the data without the need of human intervention. The field of deep network is relevant for image classification. In particular, Convolutional Neural Networks, or CNN, are widely used for the ability to take raw image data as input and to extract features that represent the interesting parts of an image in a compact way.

CNNs are presented in the chapter 4.6, meanwhile the focus is on some common feature extraction algorithms developed specifically for image processing: RGB color levels extraction, Histogram of Oriented Gradients (HOG), Scale Invariant Feature Transform (SIFT) and Texture.

2.1 RGB Color Levels

A grayscale image is made exclusively of shades of gray. Gray ranges from black to white, so from the weakest intensity that corresponds to 0, to the strongest intensity, that corresponds to 1. In Computer Vision, this type of images is stored as a 2D matrix of integers, in which pixels' values are automatically divided by 255 and scaled from 0 to 255, where 0 represents black and 255 represents white [6].

Unlike grayscale images, to store colored images is used the RGB additive color model. "RGB" stands for "Red", "Green" and "Blue". As a matter of fact, this is an additive method that uses three primary colors' shades mixing them together in various proportions to reproduce a wide spectrum of colors. So, if the 0% of each color is blended, no light is generated, creating black and this is represented with the tuple [0, 0, 0]. Vice versa, white will be denoted as [255, 255, 255] and, similarly, red will be [255, 0, 0]. In other words, the representation of a colored image in Computer Vision is a 3D tensor composed of three overlapped matrices, corresponding to the red, green and blue levels. Each of the three colors is supported by eight bits, resulting in a 24 bits color depth, called "true color". So, this provides 256 (2^8) possible values for each color with a consequent palette of 256 x 256 x 256 colors. RGB color space is widely used in literature, but note that in some case the BGR method is preferred instead of RGB: the main idea is the same, but the order of the channels is switched.

The choice of primary colors is related to the human perception of colors. Studies about the physiology of the human eye, show that the retina is sensible to different light wavelengths and that the chosen colors are able to maximize the perceived color triangle. The main aim of the RGB color model is the representation and the display of images for in televisions, video cameras, computer monitors, digital cameras, image scanners and so on.

For a better understanding of the color distribution in an image, a histogram analysis can be performed. This technique describes the number of pixels for both grayscale and colored images and can be performed on any kind of color space. The histogram analysis gives an overall idea about the intensity distribution of an image. For a grayscale image, histograms allow to determine the number of pixels that are on a scale "black pixel" (0) to "white pixel" (255). In other words, the histogram is a plot described by pixels' intensity value [0, 255] in the x axis and the corresponding number of pixels in the image (frequency) on y axis. A histogram can tell whether or not an image has been properly exposed, whether the lighting is harsh or flat, and what adjustments will work best. The X-axis indicates the range of intensity the pixel can take. This range is divided into a series of intervals called bins. For example, the whole histogram can be split into 16 bins and the value of each bin is the sum of all pixel count in it. So, 256 values are used and the range correspond to [0, 255], that is:

$$range = bin_1 \cup bin_2 \cup .... \cup bin_n = 16 = [0, 15] \cup [16, 31] \cup .... \cup [240, 255].$$

In this way, find the number of pixels for all pixel values separately is not needed, instead the number of pixels in an interval of pixel values is used. So, only 16 values are used to represent the histogram.

Colored images consist of intensity distribution of pixels where pixel value varies. Each pixel in an image has a color which has been produced by some combination of the primary colors: red, blue and green (RGB). Each of these colors can have a brightness value ranging from 0 to 255 for a digital image. A RGB histogram results when the computer scans through each of these RGB brightness values and counts how many are at each level from 0 to 255. In this case, the histogram shows the brightness distribution of each individual Red/Green/Blue color channel.

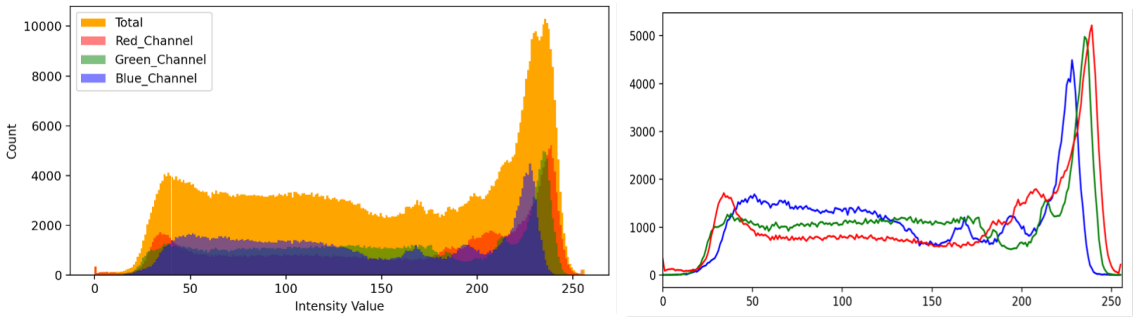


Figure 2.1: Example of distribution histogram and plot performed on a colored image using respectively Matplotlib and OpenCV libraries.

In the figure, the histogram represents the total distribution of a colored image in orange. From this graph is clear that all colors of the image have a good exposure to light. From the second plots it can be observed that all the channels have low intensities corresponding to very dark red, green and blue. Moreover, bright blue, red and green peak in the interval [200, 250]. All the colors' channels have a wide range of values.

The histogram analysis is a very simple, low-level feature extraction technique for colored image classification. As a matter of fact, in the paper by Agrawal *at al.* [16] is highlighted the benefit of using histograms: this technique is insensible to translation and rotation of the camera and seems to improve the efficiency of image classification.

Moreover, there are other feature that can be extracted to describe an image color levels: mean and variance. These two statistics can be easily computed for each channel of an RGB image. Mean values represent red, green and blue color intensities, while variance represents the contrast of red, green and blue of an image. In other words, the mean describes where the pixels are, and which color is predominant in the complete image. The variance describes how the pixel values are spread: if the variance is small, all the image colors are similar to the mean. High values of variance suggest that most of the information is not captured by the mean. To conclude, both mean and variance are two important feature that should be taken into consideration to achieve a good accuracy in image classification.

2.2 Histogram of Oriented Gradients (HOG)

This chapter introduces a feature extraction technique for images called "Histogram of Oriented Gradients", or simply "HOG", with the purpose of understanding the mathematics behind this algorithm. In 1986 the first researcher who described the concept behind HOG, but without using the term HOG, was R. K. McConnell [17]. At the Conference on Computer Vision and Pattern Recognition (CVPR) in 2005 was presented an important work on HOG descriptors by Navneet Dalal and Bill Triggs, researchers for the French National Institute for Research in Computer Science and Automation (INRIA) [18]. HOG is a feature descriptor, in particular it focuses on the structure of an object. It is used in Computer Vision and image processing to describe quantitatively an image by extracting useful information and also for object detection. This method is commonly used today and has a large number of applications ranging from face and image detection to images classification, for example in the field of autonomous vehicles or human detection. The main idea behind the Histogram of Oriented Gradients is that shapes and local objects in a photo can be described not only by the distribution of intensity gradients, but also by the edge directions. As a matter of fact, this technique counts occurrences of gradient orientation in localized portions of an image.

This type of feature descriptor converts an input image of size $64 \times 128 \times 3$, (where 64 is the width, 128 is the height and 3 is the number of channels for colored images) and gets as output a feature vector. So, the image should be resized with a width to height ratio to 1:2, preferably 64×128. That is because the image is firstly divided into cells, and the HOG is computed for each cell. Cells are small, localized regions of the image connected between themselves and are of size 8x8 pixels.

Then, HOG calculates the gradient for each of these 8x8 cells separately. Gradients are the small change in the x and y directions. Changes in x and y directions are calculated separately for each pixel in the cell. The change in the x-direction (G_x) is the absolute value of the pixel value on the left subtracted by the pixel value on the right of the selected pixel. Similarly, the change in the y-direction (G_y) is the absolute value of the pixel value below subtracted by the pixel value above the selected pixel. The results will be two new matrices: one storing gradients in the x-direction and the other in the y direction. The magnitude of the gradient is higher when there is a sharp change in pixels' intensity, such as around the edges. Using the x-direction and y-direction gradients and the Pythagoras theorem, are calculated the magnitude and the direction, that ranges from 0 to 180 degrees:

$$\textit{Total Gradient Magnitude} = \sqrt{G_x^2 + G_y^2} \tag{2.1}$$

$$\textit{Direction} = \arctan \frac{G_y}{G_x}. \tag{2.2}$$

This algorithm is repeated for each pixel value of the cell creating other two matrices: one for the gradient magnitude values and the other for the gradient direction values. So now, for every cell two matrices are known: one with the gradient magnitude values and the other with the gradient direction, or orientation. With this two information an histogram is generated for each cell. The nine bins of the histogram

represent the direction of the gradient that ranges from 0 to 180 degrees. The height of the bins represents the magnitude of the gradient. Moreover, colour images are handled calculating the derivatives for each colour channel separately, combining them to come up with one gradient magnitude and one gradient angle.

There are different methods to create histograms using gradients and orientation. The simplest way to generate the Histogram of Oriented Gradients consist in finding the orientation of each pixel and update the frequency table. In other words, each value in the gradient magnitude matrix is added in the bin that corresponds to the value in the gradient direction matrix. If the corresponding value in the gradient direction matrix is between two bins, the gradient magnitude value is divided and half of it will be added to the first bin and the other half in the second bin. Finally, the feature vector will be composed of the nine heights of the bins. The histogram can also be represented in vectorial form, where the arrows represent the directions and the length of the arrows are proportional to the magnitude of each bin. In this way, is clear in which direction the gradient magnitude is maximum.

This process is repeated in the same way for all the cells. Then, a window composed of 4 cells slides over the image concatenating the feature vectors and creating seven feature vectors of size 36 for each row and 15 feature vectors of size 36 for each column. The gradients of the image are sensitive to the overall lighting, so the feature vectors of size 36 are normalized in order to reduce lighting variation and changes in shadowing. To normalize these vectors, each of the values is divided by the square root of the sum of squares of all the values.

Finally, 7×15 feature vectors of size 36 are concatenated in the final HOG feature vector of size 3780. This is the most used method to extract the histogram of oriented gradient features. To visualize the HOG features, the histogram of each 8x8 cells is plotted in a vectorial form. In this way, the shape and the edges of the image are highlighted, especially where the change in pixels' intensity is drastic.

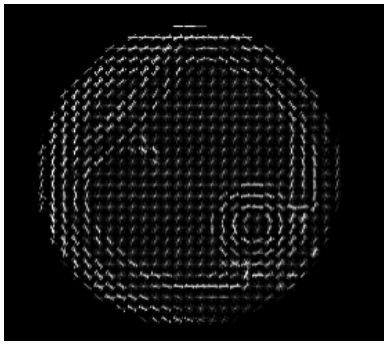


Figure 2.2: Example of HOG representation of an aluminum can cap.

In the figure 2.2, the color and the background of the flip-off are discarded. HOG representation focuses on a few pieces of information, only the shape and the edges, but this is still enough to extract the main information from the image. As a matter of fact, looking at the example, the object is easily distinguishable because it has the necessary information to be identified. That is the reason why this algorithm is used mainly for object detection. This is the key idea of a feature descriptor: use only the most important information to simplify the representation of the image.

2.3 Scale Invariant Feature Transform (SIFT)

The previous chapter 1.3 analyzes how to detect the boundaries of objects in an image, which is very useful in Computer Vision. This chapter focuses on the problem of object recognition and specifically on detecting and matching features that are fairly descriptive and unique. SIFT is an algorithm used to detect and describe local features in digital images, invented by David Lowe, a researcher in Computer Vision, in 1999 [19]. It locates specific key points and then provides them with quantitative information (so-called descriptors) which can be used for 2D object recognition. The descriptors are supposed to be invariant against various transformations that might make images look different although they represent the same object.

By considering two images of the same object (but under different conditions) is easy to notice that there is a variation in size, in orientation, in both lighting distribution and intensity and so on. An interest point detector should be able to compensate or remove these variations in order to match a feature in one image to a feature in the second image. Moreover, a point of interest detector should not only be invariant to transformation, but should also be able to consider only the most interesting and unique points in an image, discarding useless data. To identify a point of interest, it is useful to note that the local appearance around an interesting point should be rich enough in terms of brightness, size, rotation and color variations: these attributes include a certain amount of uniqueness that can be exploited for matching purposes. Secondly, a point of interest should have a well-defined position and representation, so it can be identified by a signature that is also useful for matching purposes. Moreover, it should be invariant to image rotation and scaling: in this way the point of interest should be able to produce a signature that is invariant to image rotation and scaling in addition to lightening.

A lot of edges of an object have the same kind of brightness variation, so edges are not really descriptive and unique enough to be used as interest points. Rather, after rescaling and resizing the same window region, called *blob*, of two different images, they have a comparable local appearance [20]. Blobs can be considered interesting points if they have the following characteristics:

- have an associated location in the context of the image;
- have an associated size, which is not correlated to edges or boundaries of the object, but is just a rough scale of the appearance of the blob;
- have an associated orientation, because the blob can appear rotated in different images;
- have an extractable signature that represents its appearance and is independent to size and orientation for matching purposes.

Blobs in an image can be found with a technique that uses the derivatives of images, like the tools developed for detecting edges. First of all, a Gaussian filter, see chapter 1.2.3, is used to reduce noises. Specifically, the second derivative of the Gaussian function 1.8 is applied to the image by using the convolution product in order to detect edges:

$$\nabla^2 G(x, y) \otimes I(x, y) \tag{2.3}$$

where $I(x, y)$ is the image, $G(x, y)$ is the Gaussian function and ∇^2 is the Laplacian:

$$\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}. \tag{2.4}$$

Edges are identified where the resulting function has a zero crossing. The Gaussian function is used to detect blobs in the image I and the sigma σ of the Gaussian is very important because describes the so called "scale space" associated with the image, that is different resolutions associated with the image. A change in sigma leads to find blobs at different scales or resolutions. The previous result is normalized by multiplying σ^2 to the output, in this way, blobs found with different sigmas are comparable:

$$\sigma^2 \nabla^2 G(x, y) \otimes I(x, y). \tag{2.5}$$

So, the second derivative of the Gaussian is applied for multiple sigmas, that refer to different scales. In this way, a maximum at that location of the blob is produced, and this does not depend on the size of the blob. Note that the sizes of these Gaussian functions appear to be proportional to the widths of the blobs themselves. The application of multiple sigmas results in a stack of images, which corresponds to different scales, that is the output of the second derivative of the Gaussian applied to the image, but at multiple sigma values. This is described by two parameters (x, y, σ): (x, y) for spatial coordinate and σ for the scale. Blobs are found searching the local extrema in this (x, y, σ) space. The (x^*, y^*) of the extrema is where the extrema lies and the σ^* of the extrema correspond to the scale of the blob itself. σ^* is called the "characteristic scale" of the blob and is proportional to the size of the blob. This process can be stated in the formula:

$$(x^*, y^*, \sigma^*) = \arg\max_{x,y,\sigma} |\sigma^2 \nabla^2 G(x, y) \otimes I(x, y)|. \tag{2.6}$$

The scale space can be denoted as

$$S(x, y, \sigma) = G(x, y, \sigma) \otimes I(x, y). \tag{2.7}$$

So, the previous result in equation 2.6 can be rewrite as:

$$(x^*, y^*, \sigma^*) = \arg\max_{x,y,\sigma} |\sigma^2 \nabla^2 S(x, y, \sigma)|. \tag{2.8}$$

In other words, the scale space is a stack created by filtering an image with Gaussian function with different sigma: an increase in sigma leads to higher scale, but lower resolution of the image. Sigmas to generate the scale space are selected as:

$$\sigma_k = \sigma_0 s^k \tag{2.9}$$

with $k = 0, 1, 2, 3, ...$, s is the constant multiplier and σ_0 the initial scale. To resume, blobs are identified by circles centered in (x^*, y^*) and with radius σ^*. Blobs detector can handle not only the position of the blob, but also the magnification of the blob, because an object can appear at many different scales depending on its depth from the camera.

These tools are essential to develop the SIFT detector. The SIFT detector was proposed by David Lowe and is widely used in Computer Vision because is a robust and reliable implementation of blob detection. Firstly, a stack of images is created, where is applied a Gaussian smoothing with σ, $s\sigma$, $s^2\sigma$, ..., $s^k\sigma$ in order to create a scale space $S(x, y, \sigma)$. Now the normalized Laplacian of Gaussian ($NLog = \sigma^2\nabla^2 G_\sigma$) is computed approximating it to the "Difference of Gaussian" (DoG):

$$DoG = G_{s\sigma} - G_\sigma \approx (s-1)\sigma^2\nabla^2 G_\sigma = (s-1)NLoG. \tag{2.10}$$

In this way a stack of images is found corresponding to the normalize Laplacian of Gaussian of the image computed at many different scales. Now extrema are searched in this stack of images with different method, for example with a sliding window. The interest points found are the candidates for SIFT features at many different scales. These are selected by a thresholding scheme in order to remove weak points. The remaining strong points are finally the SIFT features. Note that these points are invariant both for scale, as mentioned before, and also for orientation. Orientation invariance is made by calculating the histogram of oriented gradients, or HOG, of the blob. This technique is explained in the chapter 2.2. The peak of the HOG is called the principal orientation of the feature. When two images with different orientation are considered, the principal orientation is computed in order to undo the rotation of one to match the other image.

Now the SIFT detector computes a signature from each point of interest in order to match feature with others. For each point of interest, the SIFT detector scales and resizes the region of the point to a 16×16 pixels window. This is divided in 4×4 pixels squares and for each of them, a histogram of oriented gradients is produced in 8 directions. The histograms are concatenated in order to create a long histogram and to obtain a feature vector that represents the point of interest. This feature is insensitive to brightness and the effect of orientation, rotation and scaling are already undone by ignoring the magnitude of the gradient. So is directly used as a descriptor signature for matching to SIFT features.

Given two images, a set of points of interest of an image can be matched finding the points in the other image that minimizes the distance between the points, which are represented by vectors. Two SIFT descriptors are compared using the L_2 distance between two histograms H_1 and H_2: is essentially like comparing two arrays of data:

$$d(H_1, H_2) = \sqrt{\sum_k (H_1(k) - H_2(k))^2}. \tag{2.11}$$

L_2 distance subtracts histograms bin for bin, squared them up, and if the result is zero, $d(H_1, H_2) = 0$, then there is a perfect match. So smaller the distance metric is, better is the match.

Another metric is the normalized correlation:

$$d(H_1, H_2) = \frac{\sum_k [(H_1(k) - \overline{H_1})(H_2(k) - \overline{H_2})]}{\sqrt{\sum_k (H_1(k) - \overline{H_1})^2}\sqrt{\sum_k (H_2(k) - \overline{H_2})^2}} \tag{2.12}$$

where $\overline{H_i} = \frac{1}{N}\sum_{k=1}^{N} H_i(k)$ is the mean. If $d(H_1, H_2) = 1$, then there is a perfect match, so larger the distance metric, better is the match. This metric is used in the context of template matching.

A third possible metric is the intersection metric:

$$d(H_1, H_2) = \sum_k min(H_1(k), H_2(k)). \tag{2.13}$$

The main idea is to find bin by bin the minimum between the two histograms, like computing the overlap between two histograms. Larger the value of this metric correspond to better matches.

SIFT is an algorithm used in different fields, not only for object recognition. An example of application of the SIFT technique consists of stitching images together to create a larger image, for example a panorama. To do this, SIFT features are found in both the images and the SIFT descriptors are used to match features and to warp and align one image with the other. Another application is the construction of collages: SIFT is applied to a large collection of images and the features in the images are matched to create a single collage with a much wider field of view. SIFT is very efficient in both applications . It can be applied also to 3-dimensional objects regarded from different viewpoints, but the descriptors are not going to really match. If an image is taken from a certain angle and another image is taken from 30 degrees away, then the number of matches between these two images will have a dramatic drop. So, SIFT is reliable only for small changes in viewpoint. This is a limitation of the SIFT technique, but it still is an extremely useful and widely used detector in Computer Vision.

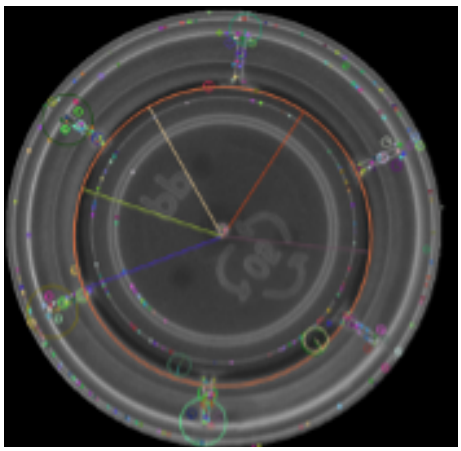


Figure 2.3: Example of SIFT performed on a gray image of a pharmaceutical ampoule cap.

2.4 Haralick Texture

Literally, the noun "texture" means "the way a surface, substance, or piece of cloth feels when you touch it, for example how rough, smooth, hard, or soft it is" [21]. In the real life, texture can be both tactile and visual: the first one refers to the tangible feeling of a surface and the second one to the perceived surface quality created by light and shadows. In Computer Vision, the texture is described as a function of spatial variation of the pixels brightness intensity and is used for object recognition, surface defect detection, pattern recognition, medical image analysis and so on.

In 1973 Robert M. Haralick, professor of computer science of the City University of New York, developed a method to describe the texture of an image [22]. This technique quantify the spatial variation of grey tone value using a so called "Gray-Level Co-occurrence Matrices", or GLCM. GLCM describes the occurrences of each gray-level pixel located in a fixed geometric position. The co-occurrence matrix captures the relationships between pairs of pixels with a matrix that codifies the occurrences of intensity levels in a pair of pixels. This matrix considers a fixed distance Q between two pixels, called the reference pixel and the neighbor pixel. For example, a *Q = (0, 1)* means that the neighbor is found by shifting zero pixels in the x direction and one pixel in the y direction, so the neighbor considered for the intensity level is the pixel at the right-hand side of the initial location. The intensity level at the observed reference and at the neighbor is computed for every pixel of an image with L gray levels. In this example the co-occurrences at the right border can't be computed because those pixels do not have a right neighbor, so they are discarded.

Then the matrix G is built as shown in equation 2.14, given a shift $Q = (\Delta x, \Delta y)$ and all pairs of intensities $i, j \in \{0, ..., L-1\}$, where L is the gray level.

$$G(i,j) = |\{(x,y)|f(x,y) = i, f(x+\Delta x, y+\Delta y) = j\}| \tag{2.14}$$

The central pixel at position (x, y) has intensity i and a pixel with a shift $(x + \Delta x, y + \Delta y)$ has an intensity j. The matrix shown in 2.14 considers as columns the levels of gray j of the neighbor pixel and as rows the levels i of the reference pixels and counts the occurrences for each pair of reference and neighbor pixels in different intensities. For example, the cell with reference pixel level equal to one, $i = 1$, and a neighbor level equal to two, $j = 2$, occurs for 4 times, so the matrix has 4 in position $(1, 2)$. So, the matrix has the number of columns and number of rows proportional to the number of gray levels L.

This matrix is often sparse, especially when L is a large value, so the image can be requantized in order to work with a smaller number of intensities. An alternative method is to normalize this matrix, as shown in 2.15, in order to interpret it as a probability of a given pair of intensities to be observed in an image.

$$p_{i,j} = \frac{G_{i,j}}{n} \tag{2.15}$$

For instance, in a normalized matrix, at position $(2, 2)$ the matrix has the value of 25%, that is the probability to random pick a pixel with an intensity value of 2 and whose neighbor has also intensity of 2.

The GLCM is used to compute descriptors or statistical measures called "Haralick descriptors". In 1973 Robert M. Haralick proposed six descriptors [22] uncorrelated with each other. The six descriptors are:

- the "maximum probability": this technique takes the strongest response of probability p in the normalized matrix in the range $[0, 1]$. This can be formulated as:

$$\max_{i,j} p_{i,j}.$$

- The "correlation": is the measure of the correlation between reference and neighbor pixels in the range $[-1, 1]$ and operates on the intensities i and j:

$$\sum_{i=1}^{L}\sum_{j=1}^{L}\frac{(i-m_r)(j-m_c)p_{i,j}}{\sigma_r\sigma_c}. \tag{2.16}$$

  Note that in equation 2.16 the mean and the variance of the rows and the columns are formulated as:

$$m_r = \sum_{i=1}^{L} i \sum_{j=1}^{L} p_{i,j},$$

$$m_c = \sum_{j=1}^{L} j \sum_{i=1}^{L} p_{i,j},$$

$$\sigma_r^2 = \sum_{i=1}^{L}(i-m_r)^2 \sum_{j=1}^{L} p_{i,j},$$

$$\sigma_c^2 = \sum_{j=1}^{L}(j-m_c) \sum_{i=1}^{L} p_{i,j}.$$

  Equation 2.16 requires standard deviations of both rows and columns different from zero, $\sigma_r \neq 0$ and $\sigma_c \neq 0$, in order to avoid zero division.

- The "contrast": considers the intensity level i and j between the pixels that appears in the GLC matrix and the actual probability of them to happen. The contrast ranges between $[0, (L-1)^2]$ and is formulated as:

$$\sum_{i=1}^{L}\sum_{j=1}^{L}(i-j)^2 p_{i,j}.$$

- The "energy": is the sum of the squares of every probability The energy ranges between $[0, 1]$ and is formulated as:

$$\sum_{i=1}^{L}\sum_{j=1}^{L} p_{i,j}^2.$$

  So the energy results one if the image is constant.

- The "homogeneity": measures the spatial auto-correlation and is formulated as:

$$\sum_{i=1}^{L}\sum_{j=1}^{L}\frac{p_{i,j}}{1+|i-j|}.$$

  Homogeneity ranges between $[0, 1]$ and results one if the matrix G is diagonal.

- The "entropy": measures the randomness of matrix G and is formulated as:

$$-\sum_{i=1}^{L}\sum_{j=1}^{L}p_{i,j}\log_2 p_{i,j}. \tag{2.17}$$

  The entropy ranges between $[0, 2\log_2 L]$ and results zero for $p_{i,j} = \mathbf{0}$.

All these descriptors are widely used because easy to interpret.

Another visual descriptor for Haralick texture analysis is the "Local Binary Patterns", or LBP. This technique was first proposed in 1990 [23] and then described in 1996 [24], however it has many variants afterwards. This method is based on the idea that texture is described by two complementary information: local spatial patterns and grey level contrast. Initially, the image is divided into cells (commonly each cell has $16x16$ pixels), and each pixel in a cell is compared to its neighbors. The P neighbor pixels lie on the circumference of a fixed radius R, so only the ones at R distance from the central pixel p, with coordinates (x_c, y_c). The LBP code developed can be formulated as:

$$LBP_{P,R} = \sum_{p=0}^{P-1} s(g_p - g_c)2^p \tag{2.18}$$

$$s(x) = \begin{cases} 1 & if\ x > 0 \\ 0 & otherwise. \end{cases} \tag{2.19}$$

In equation 2.18, g_p and g_c are the grey levels of points in the neighborhood p and the central pixel c. LBP is obtained by taking the difference between the central pixel and the remaining ones. Thanks to the threshold function 2.19, LBP will be zero for a neighbor that is below the value of the central pixel and one if it is above the value of the central pixel. Note that the number of points P to sample around the central pixel is crucial, because those sampled points will become the LBP code. For example, in the case of eight sampled pixels, $P = 8$, in a neighborhood we have $2^8 = 256$ unique codes that can be associated with a local shape patterns, so a 256-dimensional feature vector is calculated for each cell. Lastly, the feature vectors of each cell are normalized and concatenated together. Both the resulting vector and the previous Haralick descriptors can be processed using Machine Learning algorithm for texture analysis or image classification.

Chapter 3

Feature Optimization

In chapter 2 were presented some relevant feature extraction techniques used to describe the input images in an accurate and complete way. Now the goal is to build a consistent ML model for image classification with these features. The word "consistent" refers to the creation of a model able to predict the target variable with the highest degree of accuracy possible. The performance of the algorithm depends on some crucial factors: the choice of the technique, the feature used for the training, and the quality of the input data, as we know from the "garbage in, garbage out" statement. The art of finding the best feature to perform a Machine Learning algorithm is not trivial. Too many input data can lead to redundant and noisy results as well as high computational costs. Moreover, raw features not always provide an optimal implementation, but often is better to transform existing features and construct new ones to obtain a more performant model.

After the extraction of a number of features, the scientist has the responsibility to find the right methods for selecting and combining the data in order to obtain as much information as possible avoiding biases and skewness in the outcome. This is accomplished by a two-steps procedure consisting of "feature selection" and "feature engineering". The first focuses on the disposal of unnecessary features and the second focuses on techniques that conglomerate and transmute features into diverse forms to achieve more robust information for the model training. In both cases, the final aim is analogous: find the set of features that reflect and characterize as accurately as possible the considered phenomena. In these way, the number of irrelevant data is significantly reduced, but is still large enough to provide meaningful results. In addition, less redundant variables improve the generalization capability of the model, avoid overfitting and keep under control the computational complexity.

The "Occam's razor" principle, (*"novacula Occami"*), summarizes in a nutshell what has been stated so far. This pinciple was proposed by the Scholastic philosopher William of Occam (1285 - 1347) and is also known as the "law of parsimony". Occam claimed that the best explanation of an unsolved problem is always the simplest one. In other words, the solution with the fewest assumptions, or features in this case, is preferred to more complicated ones in order to avoid explainability issues. This clarifies why feature selection and engineering are crucial points of a Machine Learning model development.

3.1 Feature Selection

Phenomena can be described by a vast number of variables, but not all the available characteristics are useful for the development of the model. Sometimes, considering every variable possible can result to be redundant in computer science. "Feature selection" is the process of reducing the number of input variables selecting the features that better describe the data to achieve a robust prediction with a Machine Learning model. Feature selection has a huge impact on the performance of the model: irrelevant features can train the model based on inappropriate considerations and lead to overfitting that negatively influences the algorithm accuracy. Moreover, this technique can reduce the likelihood of biases and noises in the results as well as accelerate the training of the algorithm decreasing the complexity and the computational cost.

A lot of techniques are developed in order to discard features that do not contribute much to the resulting prediction. These features selection methods evaluate the relationship between input features and the target using statistical tests. The choice of the most appropriate measure is not trivial because their effectiveness depends on the type of the data taken as input.

Feature selection methods can be categorized in supervised and unsupervised, as explained by Ferreira *et al.* [25]. The first type finds significant features which increase the accuracy of supervised algorithm using as input labeled data. A common supervised method is called "Gini Index" or the "Information Gain" [26]. On the contrary, the second one takes as input unlabeled data, as in the case of clustering, and selects features based on measurements such as correlation or variance.

Both supervised and unsupervised methods can be taxonomically categorized into three main types:

- Filters: filter-based methods score the correlation between the target and the variables to filter out features and choose the most appropriate ones. So, it selects variables regardless of the model, although based on the intrinsic properties of the features measured by statistics such as "Pearson's correlation" [25] or "Chi-Squared test" [27].

- Wrapper methods: as for example backward or recursive feature elimination, calculate the accuracy of a specific Machine Learning model with each feature subsets and choose the set with the best result.

- Embedded: this method is inspired by both the previous types in order to handle computational cost. It selects the features iteratively during the training process of the classifier, like adding a penalty term to every iteration. An example can be "Lasso regularization" [27] or tree-based boosting algorithms [28].

3.1.1 Filter Methods

Filter methods are techniques based on the intrinsic characteristics of the features, so the selection does not depend on the Machine Learning model used. Features are ranked using statistical tests to measure the relationship with the target, regardless the performance of the model. In other words, the aim is to discard the variables with less information in order to use the others for the classification or the prediction.

These techniques are robust and can handle high-dimensional data because are efficient in time and less computationally expensive than wrapper methods. On the other hand, filter methods not always consider the relationship between the variables and select redundant variable due to multicollinearity. Some widely used filter methods are:

- The variance: is a scale-variant measure that sorts the features X_i based on their sample variance:

$$var(X_i) = \frac{1}{n}\sum_{j=1}^{n}(X_{i,j} - \bar{X}_i)^2 \tag{3.1}$$

  where $\bar{X}_i$ is the mean of feature X_i. Features with zero-variance are discarded by default because they have the same value in all samples. Moreover, also the features which do not meet a fixed variance threshold are discarded. The idea is that high variance features are more informative. This method is very simple and computationally convenient, but does not take into account the relationship between features and the target [25].

- The Mean Absolute Difference: similarly to the previous case, calculates the absolute difference from the mean value $\bar{X}_i$. The MAD is scale variant, like the variance in the formula 3.1, but has no square, as shown in the formula 3.2:

$$MAD_i = \frac{1}{n}\sum_{j=1}^{n}|X_{i,j} - \bar{X}_i|. \tag{3.2}$$

  A high MAD leads to a high discriminatory power [25].

- Dispersion ratio: is a dispersion measure that assess relevance through dispersion, regardless of the class label. This ratio uses the arithmetic mean (AM) and the geometric mean (GM) for feature X_i, that are formulated as:

$$AM_i = \bar{X}_i = \frac{1}{n}\sum_{j=1}^{n} X_{ij}$$

$$GM_i = (\prod_{j=1}^{n} X_{ij})^{\frac{1}{n}}.$$

  The Dispartion ratio is formulated as:

$$DR_i = \frac{AM_i}{GM_i}$$

and since $AM_i > GM_i$, than it ranges from 1 to $+\infty$ [25]. A high Dispersion ratio indicates a more relevant feature. A DR close to 1 means that the feature can't discriminate properly.

- Fisher's score: is a supervised feature selection algorithm that measures the discriminating power of the variables. In other words, selects the features whose values are similar within the same class and dissimilar within different classes [27]. The Fisher's score for feature i can be formulated as:

$$FiR_i = \frac{\sum_{j=1}^{c} n_j(\bar{X_{ij}} - \bar{X_i})^2}{\sum_{j=1}^{c} n_j \sqrt{var(X_{ij})}}$$

where n_j is the number of samples in class j, $\bar{X_i}$ and is the mean of feature i for samples in class j and $var(X_i)$ is the variance of feature i for samples in class j. The final rank is based on the Fischer's score of features in descending order [25].

- Chi-Square Test: is a statistical test for categorical features based on the Chi-Square distribution and is useful to decide if a variable is related or not to the target. This method calculates the Chi-Square between each feature and the target and selects the desired number of features with the best Chi-Squared scores.

$$\chi^2_{DF} = \sum_{i=1}^{m}\sum_{j=1}^{n} \frac{(O_{i,j} - E_{i,j})^2}{E_{i,j}} \tag{3.3}$$

The Chi-Square statistic is formulated as showed in 3.3 where O means observed or actual frequency, so the number of observations contained in the classes i and j. E means expected frequency, that is the number of expected observations that belong in the classes i and j if there was no relationship between the feature and the target. "DF" stands for "Degrees of Freedom" and is the number of categories for the classification minus 1. The null hypothesis assumes that there is no association between the two variables and can be rejected when the Chi-Square p-value is greater than the alpha level α, that is usually the 5%. If this happens, the variables can be chosen for the model. On the contrary, a small Chi-Square value means that the feature is independent with the target and it is uninforming for the classification [27].

- Pearson's Correlation: measures the linear dependency between two continuous variables X and Y. Pearson's correlation is the ratio between the covariance of X and Y and the product of their standard deviations σ_X and σ_Y, as shown in the formula 3.4. The result ranges from -1 to +1, where +1 ideally represents the perfect correlation.

$$\rho_{X,Y} = \frac{cov(X,Y)}{\sigma_X \sigma_Y} \tag{3.4}$$

This ratio considers only one type of relationship, the linear one, ignoring all the others. The Pearson's correlation describes how strong is the linear relation between a variable, or a set of variables, and the target. Furthermore,

this measure takes into consideration if the variables are correlated among themselves. This is crucial because if two variables are correlated, the model do not need both. If one variable is predictable from another one, include both will add uninformative insights to the model [25].

- The Information Gain: In information theory, the word "information" refers to how surprising an event is, and events have more information if they have a low likelihood of happening. The Information Gain score is calculated for each variable to determine if a feature contains suitable information for a split in two different classes and is based on the reduction in entropy after the dataset partitioning. The entropy quantifies the probability distribution of observations belonging to one class or another in order to describe how much information a splitting variable has. Information Gain is formulated as:

$$IG(T, a) = H(T) - H(T|a) \tag{3.5}$$

where a is the set of attributes partitioned by a random variable T and $H(T|a)$ is the entropy of the variable T given the value of attribute a. A high value of IG indicates a high information content and a low entropy. When two events are equally probable, so if the dataset can be split in 50-50 samples in two classes, they have the largest entropy and the lowest value of information. In this way, the Information Gain ranks the features in order to distinguish relevant from uninformative features [29].

These are the most used filter methods, but there are many others as presented by Huan Liu *et al.* [27]. All the filter methods consist of two steps. Firstly, these measures evaluate the importance of the features in order to rank them. The main idea is to select the most informative features accessing their correlation with the target variable relying on the characteristics of data. Moreover, they are independent from the machine learning model used for the classification, so they do not require lot of computational time. In the second step, redundant features are filtered out on the basis of statistical tests. In this way, the efficiency and accuracy of the classification model is improved.

3.1.2 Wrapper Methods

The Wrapper methodology considers the features selection as a search problem. Features are combined in different sets and each set is evaluated with a predictive model and is scored in order to be compared to the others. In other words, this technique is a search algorithm that use model performance to select the best set of features [30]. Due to this, the accuracy and the efficiency of the Wrapper method depends on the classifier used, that could be for example a linear regression or a random forest.

Moreover, this type of algorithms can be "greedy" or "non-greedy" [31]. An approach is considered greedy when it evaluates all the possible features combinations searching in the direction that seems best in order to make a locally optimal choice at every iteration. A non-greedy algorithm can reconsider features that were previously discarded. In this way, it would not consider only some local optima.

Different methods are used in order to create various feature set:

- Forward Selection: is a greedy approach that starts with no feature and then adds iteratively feature in the model which best improves the performance. It stops adding features when the model does not improve anymore. "Sequential Forward Selection", or SFS, is a greedy search algorithm of this type and is often used for linear regression.

- Backward Elimination: is a greedy approach that starts selecting all the feature available and then removes iteratively the features relying on a performance improvement of the model until the best accuracy is achieved. One of the most widely used algorithm of this type is the "Recursive Feature Elimination", or RFE, as explained by Kuhn *at al.* [30]. This is a greedy optimization algorithm that ranks the features based on the order of their elimination. In this way, the least important features are discarded from the feature set.

- Step-wise selection: is a combination of forward selection and backward elimination, so is a bi-directional non-greedy approach. Basically, this technique adds variables that has been removed back into the model, and vice versa, at any iteration.

- Exhaustive Feature Selection: evaluates every possible combination of features, so is a brute-force approach. This is the most computationally expensive and slowest wrapper algorithm.

Wrappers and filter methods, explained in chapter 3.1.1, differ in many different aspects. The wrapper method trains a model in order to access the variables importance. So, they are very accurate in finding the best subset of features, but much more computationally expensive than filter techniques. On the other hand, Filter methods are faster because measure the relevance of a features subset by accessing their correlation with the target using statistical tests.

3.1.3 Embedded Methods

Embedded methods is a trade-off between filter and wrapper methods, that are described in chapters 3.1.1 and 3.1.2, because they combine the qualities of both [27]. These techniques are faster than wrapper methods because they don't train iteratively different subsets, but select features within the training of the classifier. In other words, the feature selection procedure is naturally incorporated as part of the training process [32]. So, the feature selection and the algorithm training are performed in parallel. In this way, the algorithm is more efficient and less computationally expensive than wrapper methods.

The most widely used Embedded methods are:

- LASSO and Ridge regressions: performs respectively L_1 and L_2 Regularization adding a penalty term to the cost function in order to reduce the degree of freedom and penalize more complex models. So, these are a generalization of the linear models and are used for small datasets, where each observation has a large impact on the model, in order to avoid overfitting. The cost functions of LASSO and Ridge regressions are formulated as:

$$L(X, \theta) = \min_{\theta} \frac{1}{N} ||y - X\theta|| + \lambda ||\theta||_q$$

  where θ is the coefficient vector and X is the feature vector of the regression. The L_1 penalty term is equivalent to the absolute value of the magnitude of coefficients, that is the case of $q = 1$. The L_2 penalty term is equivalent to the square of the magnitude of coefficients, the case of $q = 2$. λ is a non-negative complexity parameter that controls the regularization strength, so a higher λ will penalize more the outliers. So, adding a regularization term, the problem becomes a minimization of the regularized cost. These methods perform both variable selection and regularization at the same time. If a coefficient results to be zero then the relative feature is considered uninformative and is discarded [27].

- Tree-based selection: is a simple method based on a Decision Tree or a Random Forest, models explained in chapter 4. Variables are selected when the model uses them in a split and the discarded features are the ones ignored for splitting. In order words, features are ranked by the splitting method of the tree-based algorithm, that is the "Gini index" or "Information Gain", previously explained in chapter 3.1.1 and shown in the formula 3.5. The lower the value of the Information Gain, the more important the feature is. Sometimes this approach can leads to biased results because tends to inflate the importance of continuous features or high-cardinality categorical variables. Often for feature selection is used a "XGBoost" that is a gradient boosting decision tree algorithm. The word "XGBoost" stands for "eXtreme Gradient Boosting". The boosting techniques build strong classifier using several weak classifiers in series. The final result is a weighted average of the various results, with higher weights for the stronger classifiers. Moreover, the data distribution is randomized before the classification is performed. In this way, the model can control both data bias and variance and is much more efficient [28].

- Regression coefficients: a simple way to access feature importance is by examining directly the coefficients of a regression model, after standardizing all the feature on the same scale. So, the absolute values of the equation coefficients are used to rank the features. The main idea is, if the coefficient is large, then the feature will influence the prediction. The uninformative features that have coefficient zero are discarded because they have no influence on the prediction.

- Principal Component Analysis: is a dimensionality-reduction technique. PCA is useful to reduce the dimensionality of large data sets, but maintaining most of the information. This technique is able to select a number of important features. PCA is used for feature selection because it can access features importance, but not directly. Firstly, the data are scaled and fitted in the model. Then the PCA returns the principal components that can explain better the variance of the data. The most important component is the first principal component, because it explains a big percentage of the variance of the dataset. The second principal component is the second-best that describes the variance, but should be uncorrelated with the first principal component. The importance of each original feature is evaluated by the analysis of the covariance matrix eigenvectors. In this way, PCA can reduce the dimensionality of the original dataset, without reducing the information contained originally [33].

3.2 Feature Engineering

"Feature engineering" refers to the manipulation and the creation of features in order to improve the quality of a Machine Learning model. The development of new valuable input is based on the analysis of the potential information gain to describe as best as possible the initial data. The new features are useful to understand the characteristics of the data. In other words, feature engineering is a sort of mathematical optimization that allows the transformation of existing features for better analysis. This technique is different from the feature selection that focuses on removing redundant features, as explained by Kuhn *et al.* [31].

Features, such as dates, often cannot be used as input in their raw form, but they have to be transformed into a numerical representation. Moreover, features aggregation or manipulation can lead to more appropriate information contained within the features. For example, the variable "age" can be more effective if is aggregated in ranges.

Lastly, the combination of two or more features can increase the statistical relationship meaning between the target and the model input. For instance, the ratio height-weight describes better a human body than considering only one of these characteristics.

Feature engineering can be of two classes:

- The manual feature engineering method uses features importance measurements to evaluate the correlation with the target. This requires a good intuition and experience of the scientist to find skewness or bias in the data and is very time consuming.

- The automated feature engineering method is based on algorithms that automatically combine and find the best feature sets.

Feature engineering is not trivial, but is an important step to achieve a good performance. In Machine Learning, the most accurate the model is, the most useful insights are produced.

3.2.1 Manual Feature Engineering

Manual feature engineering is performed by the analyst in person, so is influenced by human interpretation errors. First, the scientist performs an exporatory analysis in order to achieve a good background knowledge of the dataset, especially understanding how the variables are correlated to the target. Then, are performed techniques intuitively chosen from the data inspection. For example, a variable can result skewed towards a particular value, so proper choices can be made to provide better information for the model. Different techniques can be used to transform features, some specific methods are more performant with some algorithms, but others work well in different situations.

Often a simple function is applied to a feature in order to transform it according to another representation, in order to make it more readable and easier to visualize. Dates and timetables are always manipulated by the machine learning algorithms that can handle only specific formats.

Moreover, often the features contain string formats that machine learning algorithms cannot comprehend. Splitting the features is a good way to extract the utilizable part without losing information. A lot of different splitting techniques are developed in order to face multiple cases depending on the characteristics of the specific feature.

A widely used mathematical function is the logarithm. The log transform is performed on skewed data in order to approximatively turn them into a normal distribution, as explained by Zheng *at al.* [34]. Less-skewed data are easier to handle by machine learning algorithm and can provide more truthful information. Moreover, this function can manage outliers decreasing their influences on the final prediction thanks to the normalization of magnitude differences. Note that this technique is limited only for positive values and often is better to add 1 to the data before the transformation to ensure a positive output.

A common manual feature engineer technique to deal with categorical data without losing any information is the "One-hot encoding", as explained by Zheng *at al.* [34]. If a categorical feature has N possible categories, the encode feature will spread them and map them in *N-1* new features, because the N category can be deducted by the others. Every sample of each category is represented with only two values: 0 and 1. In this way, the categorical feature is represented numerically in order to be more readable by the algorithm.

Numerical features can be difficult to compare because they are often centered on different ranges. Scaling is a good solution to re-center the data and to translate values changing the extremes, as explained by Kuhn *et al.* [30]. To convert the values, a linear formula is applied to the data:

$$y = mx + b$$

where the coefficient m controls the slope and b shifts the data. Scaling is mandatory only for distance based algorithms, like K-Nearest Neighbor explained in chapter 4.3. However, this technique is widely used in numerous scenarios, for example for converting Celsius to Fahrenheit or meters to feet, but also to fit values in a $[0, 1]$ or $[-1, 1]$ range.

The most used scaling techniques are:

- The normalization: also called "min-max normalization", scales values in the fixed range $[0, 1]$ using the formula 3.6. Note that, this transformation cannot reduce the effect of outliers because does not influence the data distribution.

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}} \tag{3.6}$$

- The standardization: also known as "z-score normalization", scales data considering the standard deviation in order to limit the outliers effect using the formula 3.7. So, this technique does not consider a fixed range a priori. In this way, the distribution's mean and the variance result equal to 0 and 1 respectivrly[34].

$$Z = \frac{X - mean(X)}{std(X)} \tag{3.7}$$

How the features are manipulated can impact significantly on model performance, as explained by Kuhn *et al.* [30]. Also combining more features can be a good choice, but is not trivial and is effective only if the analyst has a deep understanding of the problem. In other words, manual feature engineering is crucial to achieve a good performance, but is influenced by human errors.

3.2.2 Automated Feature Engineering

The manual feature engineering is a traditional approach and is very time-consuming because every feature is manipulated at a time. Moreover, is based by the human intuition, so is more prone to errors. Automated feature engineering solves these problems and automatically synthesizes more features at the same time, so is significantly faster. In this way, is not necessary to rewrite the code for every feature. This method can extract informative features with a standardized process. In this way, the algorithm is more efficient as well as repeatable.

The automatization of the feature engineering process can be made by different tools, some examples are:

- AutoFeat: means "Automatic Feature" and is an algorithm that automatically manipulate the features. This technique can be used for both classification and regression models and can handle categorical features using the One-hot encoding, previously explained in chapter 3.2.1, but cannot handle missing values. First, the AutoFeat transforms the feature vector using non-linear transformations, then combines the features together in order to gather most of the information. Lastly, the algorithm chooses the optima features using a LASSO regression approach, as explained by Horn *at al.* [35].

- TSFRESH: means "Time Series FeatuRe Extraction on basis of Scalable Hypothesis tests" and is an open source Python tool for time series classification and regression. This algorithm extracts a lot of characteristics in parallel from time series data, as explained by Christ *at al.* [36]. Moreover, TSFRESH can evaluate the explanatory power and the importance of these characteristics.

- OneBM: stands for "One Button Machine" and is an algorithm that automates feature discovery in relational databases based on depth-first search, as explained by Hoang Thanh Lam *at al.* [37]. OneBM automatically joins raw data tables and automatically identifies data types. In this way, it applies predefined feature engineering techniques to efficiently extract relevant features, indeed outperforms in Kaggle competitions.

- ExploreKit: is a method for automated feature generation, as explained by Katz *at al.* [38]. This technique combines the original features in order to generate new ones that maximize the performance of the model. Moreover, it keeps only the more informative features to avoid the exponential growth of the algorithm complexity using feature selection approach based on Machine Learning. The main idea is that the most informative features are the result of manipulating basic ones, so some common operators are developed to transform and combine the original features and create new ones.

Chapter 4

Classification Algorithm

Classification is a technique for the categorization of structured or unstructured data into a fixed number of categories. Literally, the word "classification" means "the act or process of putting people or things into a group or class" [39]. The algorithms that perform this process on the input data are called "classifiers". A "classification model" uses the classifiers to predict the membership category of a new sample. Models can be divided into linear, such as Logistic Regression and Support Vector Machine, and non-linear, such as Decision Tree, Random Forest and K-Nearest Neighbors. This chapter will referred especially to the book by Szeliski *et al.* [11] and the book by Gèron *et al.* [40].

Moreover, the classification can be "binary", "multi-class" or "multi-label", as explained. A "binary" classification deals with just two classes, for example male/female, yes/no or black/white. On the other hand, a classification can be defined as "multi-class" if can handle more than two categories, but every sample can belong just to one category, for example an animal can be a cat, a dog or a bird, but not a cat and a dog simultaneously. In "multi-label" classification each sample can be part of more than one category, for example an article can talk about more than one topic. This chapter will focus on multi-class supervised algorithms.

"Supervised learning" is a method that uses the "Ground Truth" during the training of the algorithm. The "Ground Truth" is also called "empirical evidence" and deals with the comparison of the results against the real world. In other words, the data taken as input are labeled, so contain the most important information: the corresponding category. Finally, the algorithm will classify new samples during the validation process, without knowing the class of membership a priori, as explained. On the contrary, "unsupervised classification" is a fully automated method that takes as input a huge number of samples and classifies them into some categories based on natural groupings present in the original data, so without knowing the Ground Truth. So, this type of algorithms analyzes unlabeled data and searches patterns in order to cluster them automatically.

Computer Vision is based on the image classification task because is the fundamental problem that solves the other issues, such as object detection. Image classification is a classification task that involves a large set of images representing an object. The final aim is to build an algorithm that exploits the features extracted, as explained in chapter 2, in order to classify the object in the right class with a

certain probability. Nowadays, image classification is a hot topic and is involved in many areas of healthcare and 4.0 industry, for example vision systems are exploited for medical diagnosis, anomaly detection or traffic control.

Classifying images is a complex task when performed by the human brain and is certainly even more challenging when machines are involved. Many classification algorithms have been developed to solve this issue, in this chapter some of these will be described, such as:

- Decision Tree classifier: is a tree structure flowchart composed of nodes and branches. Each node applies a test on an attribute and each branch represents an outcome of the test. Finally, each leaf node represents a class label. Decision trees classify instances by sorting them down the tree from the root to some leaf node, which provides the classification of the instance.

- Random Forest: is a supervised Machine Learning algorithm that builds a number of Decision Trees on different random subsets of the dataset and attributes. Finally, it takes the trees majority vote for classification or average for regression.

- K-Nearest Neighbors: or KNN, is an approach to data classification that estimates how likely a data point is to be a member of one group amongst many, depending on what group the data points nearest to it are in.

- Support Vector Machine: or SVM, is a supervised learning algorithm based on the idea of finding a hyperplane that best separates the features into different domains. A SVM's kernel, the matrix that converts the input data space into a higher-dimensional space, can be linear or polynomial (3^{rd} or higher degree). The degree parameter controls the flexibility of the decision boundary. Higher degree kernels yield a more flexible decision boundary. The idea is to map data points to high dimensional space to gain mutual linear separation between every two classes.

- Multiple Classifier: is an ensemble classifier that builds a set of learners and aggregate them to end up with a model that performs significantly better than every other single model.

- Convolutional Neural Networks: or CNN, is a class of Artificial Neural Network and is used to analyze visual imagery, in fact the first layer takes as input a 3D volume. The first convolutional layer is responsible for capturing the Low-Level features such as edges, colour, gradient orientation, etc. With added layers, the architecture adapts to the High-Level features, returning a network which has the wholesome understanding of images in the dataset. The role of the CNNs is to reduce the images into a form which is easier to process, without losing features which are critical for getting a good prediction.

- Transformers: are efficient Neural Network architectures that exploit the so called "Attention Mechanisms". This technique is widely used in the fields of Natural Language Processing (NLP) and Computer Vision.

4.1 Decision Tree

One of the most popular supervised Machine Learning algorithms is the Decision Tree, or DT. This technique involves a tree structure flowchart made up of a set of n nodes, or vertices, connected by *n-1* directed branches, or arcs. In graph theory, a "tree" is a connected acyclic graph. Acyclic means that the first and the last nodes cannot be connected, that is the reason why a tree cannot have more than *n-1* branches. The first node is called "root" and the nodes without child nodes are called "leaf" nodes and, graphically, are located at the end of the tree.

In a Decision Tree each node performs a test on the feature to create a split. The aim is to find a hierarchical structure in order to explain how input data correspond to different outcomes by traversing the tree. The internal nodes can be of three types: boolean, if the node has only two child nodes, nominal, if the test involves a finite set of possible output values because is based on nominal attributes, and continuous, when numerical attributes are split based on a fixed threshold.

Decision Trees can be used for both regression and classification problems. Regression Decision trees have nominal attributes and aim at predicting continuous values. Decision Tree Classifiers deal with the classification of discrete values, so the final purpose is to predict the membership category of an input sample. This chapter will focus on the second task, because is propaedeutic for image classification tasks.

In order to predict successfully a class, a DT splits the input set into subsets based on an attribute value test. In this way, DTs implicitly perform feature selection. Based on the characteristics of the initial training set, the model learns a series of questions to ask in order to understand the class of the sample in question.

The metric used to decide when to split the data is the "Information Gain", or IG. The IG is the objective function to maximize in order to divide the dataset into the most informative features. The concept of "Information Gain" was already cited in chapter 3.1.1 and, in this context, the equation 3.5 can be reformulated as:

$$IG(D_p, a) = H(D_p) - \sum_{j=1}^{m} \frac{N_j}{N_p} H(D_j) \tag{4.1}$$

where a is the attribute on which to perform the split, D_p and D_j are the dataset of the parent and of the j^{th} child node respectively. H is our measure of impurity, N_p and N_j are respectively the total number of samples at the parent node and in the j^{th} node. In practice, the IG is the difference between the parent node and the child node impurity. The lower the impurity of the child node, the higher the IG.

In order to reduce the complexity, often are implemented binary trees in which each parent node is split into at most 2 child nodes (left and right), so the formula becomes:

$$IG(D_p, a) = H(D_p) - \frac{N_{left}}{N_p} H(D_{left}) - \frac{N_{right}}{N_p} H(D_{right}).$$

One of the functions H that measures the impurity is the "entropy". The concept of "entropy" was already cited in chapter 2.4 and formulated as shown in equation

2.17. In the case of a Decision Tree, the entropy can be reformulated as:

$$H(t) = -\sum_{i=1}^{C} p(i|t) \log_2 p(i|t)$$

where $p(i|t)$ represents the proportion of samples that belong to class C for a particular node t. Entropy is measured in bits, so if a subset is unequally divided, the entropy ranges from 0 to 1, extremes excluded. A zero-bit entropy corresponds to a 100% pure subset, so there is no information missing and no uncertainty. For example, in a binary scenario, the entropy is zero if $p(i = 1|t) = 1$ and $p(i = 0|t) = 0$. A one-bit entropy represents a subset equally divided, so 50-50 division like $p(i = 1|t) = 0.5$ and $p(i = 0|t) = 0.5$. Basically, the Information Gain questions if a particular split could increase the entropy or no, indeed the IG formula 4.1 measures the difference in entropy before and after the split.

Another functions H that measures the impurity is the "Gini impurity". Both entropy and Gini impurity lead to very similar results. The Gini impurity is defined as :

$$H(t) = 1 - \sum_{i=1}^{C} p(i|t)^2.$$

Again, the highest value is reached when the classes are perfectly mixed [40]. Gini impurity is computationally faster, but entropy tends to produce slightly more balanced trees.

To resume, the algorithm selects an attribute for the splitting that has the least amount of information missing, and therefore a greater certainty, that means the lowest amount of bits. In other words, the chosen attribute has the lowest level of entropy and, so, the greater purity. This partitioning process is repeated recursively on each derived subset. The process will end when splitting does not increase anymore the prediction accuracy. So, a Decision Tree is constructed top-to-bottom by splitting the samples into more specific and lower entropy subsets.

Unlike others Machine Learning algorithms, Decision Trees do not process feature vectors all at once, but analyze single attributes to decide which to use [11]. In this way, this is a great tool to visualize the question and to understand which are the most valuable options for facing the problem. Moreover, Decision Trees can handle high-dimensional data, both numerical and categorical, ensuring a good accuracy for the prediction. That is because features nonlinear relationships do not affect the performance.

On the other hand, Decision Trees often experience several issues. DTs can become over-complex and, as such, are not able to generalize well the algorithm. The high risk of overfitting is because they are focused on local optima. Moreover, a small adjustment in the upper part of the tree can lead to completely different results. So, they are unstable because they can change very quickly their structure for a small variation in the data. This variance problem can be solved using methods like "bagging" and "boosting". "Bagging" stands for "Bootstrap Aggregation" and faces the variance issue training in parallel multiple trees with subset of data sampled randomly. The result is an ensemble of different models, and all the prediction are averaged to ensure a robust decision. "Boosting" methods are used to build strong

classifier using several weak classifiers in series. So, trees are fitted consecutively on random samples and aim to improve the accuracy at every step increasing the weights of the misclassified samples. A type of boosting method, called "XGBoost" were cited in chapter 3.1.3.

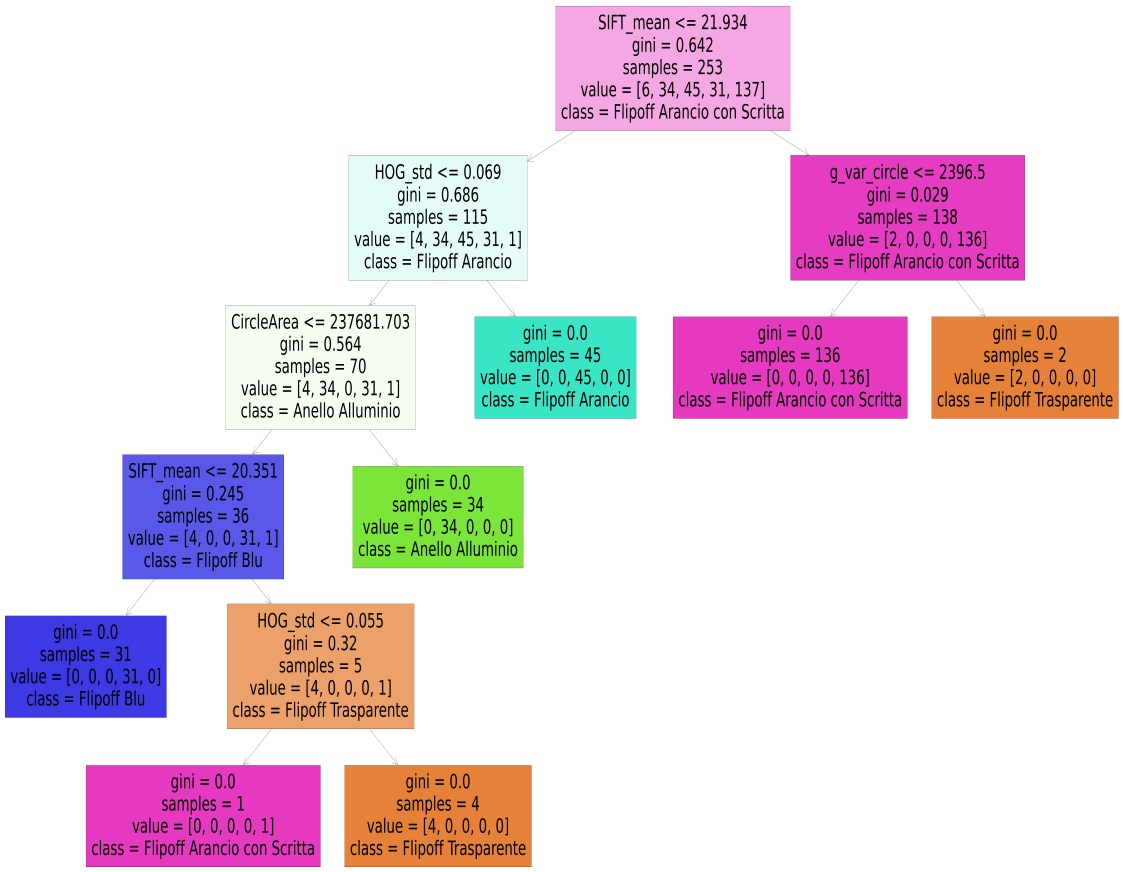


Figure 4.1: Example of Decision Tree performed on images of a pharmaceutical ampoule cap.

4.2 Random Forest

Another widely diffused supervised Machine Learning algorithm is the Random Forest, or RF. The term "Random Forests" was first coined by Tin Kam Ho in 1995 [41] during the "3^{rd} International Conference on Document Analysis and Recognition", in Montreal. The word "Forest" in graph theory refers to a disconnected acyclic graph. A RF is a disjoined group of decision trees built on different random subsets of the dataset and attributes.

This algorithm can handle both regression and classification problems. A Random Forest deals with categorical variables and ends up with the trees majority vote in the classification case. For the regression case, a RF handles continuous variables and takes as final result the average of the trees predictions. The Random Forest algorithm performs better with classification problems and is more prone to errors in the regression case because of the final prediction decision method. However, Random Forests perform well with both categorical and numerical data without the necessity of scaling or transforming the variables.

Random Forests can be seen as an extension over Bagging [40], so they are more robust than single Decision Tree because they control the variance issues that usually affect DTs. Random Forest algorithm goes one step further over Bagging (chapter 4.1): not only considers a random subset of the initial dataset, but also selects randomly the features to use for every single tree. Because of this characteristic, RFs are considered an Ensemble method, indeed, a collection of trees is often called an "ensemble".

In other words, in order to implement a Random Forest, firstly the training data set is taken randomly with replacement. Then a subset of features is selected randomly to grown a Decision Tree (chapter 4.1). Finally, this process is repeated a fixed number of times T to build a fixed number of trees T and each tree makes a slightly different decisions $f_t(x)$, so the final prediction is the aggregation of each tree result [11]. The final prediction p for a Random Forest can be formulated as:

$$p = \frac{1}{T} \sum_{t=1}^{T} f_t(x)$$

where T is the total number of trees in the Random Forest, f_t represents a single tree, so a classification or a regression on a random sample of data, and x is the prediction of each class for a single tree.

This type of ML algorithm is often chosen for many advantages. RF can handle high dimensional data, outliers and missing values as well as both linear and non-linear relationships in the data. For these reasons, Random Forests are often considered for the image classification task. Moreover, every single tree considers only a subset of the available features. In this way, the trees result to be simple, not very deep and, as such, less prone to overfitting than Decision Tree algorithm. Thanks to that, RFs ensure a better accuracy than the one obtained with a single tree.

On the other hand, Random Forests are not easy models to interpret and to control. In addition, they can be time consuming and computationally intensive for large datasets training.

Random Forest can be applied to different fields, in addition to image classification task, including:

- Banking: for decision making in problems of clients' creditworthiness, fraud detection or stocks behavior identification.
- Health care: for patients diagnosis assessing their previous medical history.
- E-commerce or streaming platforms: to analyze customers behavior and to predict their preferences for product recommendation and price optimization.

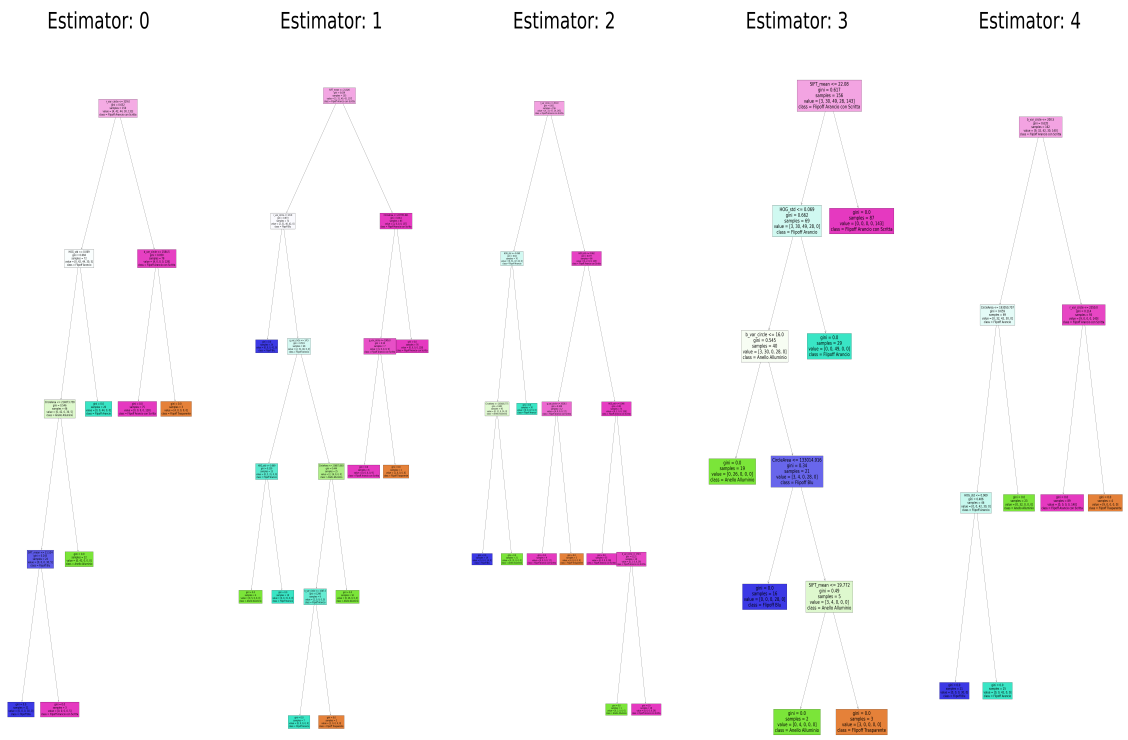


Figure 4.2: Example of Random Forest with 5 trees performed on images of a pharmaceutical ampoule cap.

4.3 K-Nearest Neighbors (KNN)

The "K-Nearest Neighbors" algorithm, or KNN, is a non-parametric supervised learning algorithm that estimates how likely a data point is to be a member of one group amongst many examining the nearest data. A "non-parametric" statistic does not consider the underlying assumptions on the shape of the probability distribution. This topic was first explored by the statisticians Fix and Hodges in 1951 [42] and expanded by Cover in 1992 [43].

The word "neighbors" means that two data points are near to each other, and their distance can be measured by different types of metrics. The main idea is to exploit the distance between two points in order to access their similarity. In other words, KNN can learn from labeled input data how to classify new unknown data.

The KNN can be used for both regression and classification tasks. The main distinction is that classification is used for discrete values, while regression is used with continuous ones. In the regression case, the final result is the average of the property values of k-nearest neighbor points. For the classification, the output results to be a class of membership estimated by a k-neighbors majority voting scheme. So, the final result is the most frequent label near a particular data point. The K-Nearest Neighbors algorithm is widely used especially for classification problem.

In the word "K-Nearest Neighbors", "K" indicates the number of closest neighbors to take into consideration for the classification of a specific point. For example, in the figure 4.3, if $K = 1$ the new point, represented by a star, will be classified as the single nearest neighbor. Increasing the value of K changes the final output from the red to the blue class. The choice of the K value is not trivial: a too high or too low K can lead to very simple or complex model that results in, respectively, underfitting and overfitting. A good trade-off is fundamental for a robust model. This choice is extremely influenced by the input data. Higher values of K suit better dataset with a lot of outliers. That's because higher values of K tend to average the prediction values over a larger neighborhood area. Frequently, cross-validation techniques are used to choose the optimal value for K.

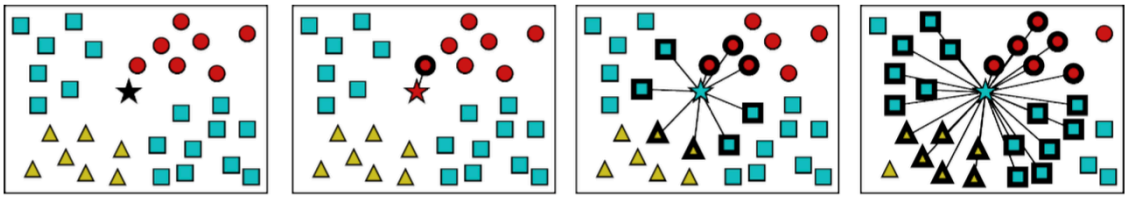


Figure 4.3: KNN classification: a graphical example for various values of K, precisely for k = 1, 9, and 25. Image from book [11].

Another hot topic of the KNN algorithm is the metric to calculate the distance, or dissimilarity, between two points in a multi-dimensional space. The choice of a specific metric can slightly change the resulting decision boundaries. Some examples of suitable metrics are:

- Euclidean distance: is most used default metric and is limited to vectors with real values. It calculates the distance as a straight line between the two points

in the Euclidean space and is formulated as:

$$d(x, y) = \sqrt{\sum_{i=1}^{n}(y_i - x_i)^2}.$$

- Manhattan distance: is another popular distance metric. It calculates the absolute value between the Cartesian coordinates of two points. This distance is also known as "city block distance" and comes from how the distance between two addresses through the city streets is measured. This metric is formulated as:

$$d(x, y) = \sum_{i=1}^{n} |y_i - x_i|.$$

- Minkowski distance: is a metric for real-valued vector spaces and is formulated as:

$$d(x, y) = (\sum_{i=1}^{n} |y_i - x_i|^p)^{\frac{1}{p}}.$$

  From the formula is easy to note that the Minkowski distance is the generalized form of the Euclidean, when $p = 2$, and Manhattan distance metrics, when $p = 1$. Moreover, this metric satisfies the conditions of non-negativity $d(x, y) \geq 0$, identity $d(x, y) = 0$ *if and only if* $x == y$, symmetry $d(x, y) = d(y, x)$ and triangle inequality $d(x, y) + d(y, z) \geq d(x, z)$.

- Jaccard coefficient: is a metric that measures the similarity of two datasets, A and B, and is formulated as:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}.$$

  The formula represents the size of the intersection divided by the size of the union of two label sets. Is preferable to don't use this metric for small datasets because missing observations can drastically influence the result.

- Cosine distance: is used to calculate similarity, that is $\cos\theta$, between two vectors and the distance metric is formulated as:

$$d(x, y) = 1 - \cos\theta = 1 - \frac{\vec{a} \cdot \vec{b}}{||\vec{a}|| \cdot ||\vec{b}||}.$$

  This formula exploits the cosine of the angle between two vectors to determine whether two vectors are pointing in the same direction. The Cosine distance ranges from 0 to 1. If the distance tends to zero, the vectors are very similar. This is not a conventional metric for KNN, but is useful to find out new insight from different point of views.

- Hamming distance: is considered an overlap metric because calculates the similarity between two binary data strings of equal length comparing the bit

positions. The Hamming distance results in the number of attributes that are different between two vectors and is formulated as:

$$d(x, y) = \sum_{i=1}^{n} |y_i - x_i|$$

where $d(x_i, y_i) = 0$ if $x_i = y_i$ and $d(x_i, y_i) = 1$ if $x_i \neq y_i$.

The distance between points is the central concept for KNN classification, but often data has different scales, so is important to normalize the features in order to achieve a high accuracy.

KNN is a popular algorithm thanks to the easy implementation and the good accuracy. Moreover, is easy to adapt for new input samples and needs only the K value and a metric as hyperparameters. On the other hand, for large dataset is very time consuming. It is not very efficient in term of memory and data storage management and that can compromise the algorithm performance. Furthermore, KNN does not handle well high dimensional data during the training process. Indeed, high dimensional input negatively affects the algorithm performance. This phenomenon is called "curse of dimensionality". To reduce this risk, feature selection and dimensionality reduction techniques are exploited.

In some cases, some further techniques are used to increase the KNN algorithm accuracy. For example, assign weights to the neighborhood points results to be useful. In this way, the closest points influence more the final result. An efficient technique consists in assigning at each neighbor point a weight corresponding to $\frac{1}{d}$, where d is the distance between the new point and the neighbor point and is calculated with one of the metrics discussed above.

The KNN algorithm is exploited in a vast range of purposes, mainly for the classification task, such as:

- image classification: KNN is widely used for image classification. For example, the paper written by Zhuang *at al.* [44] discusses how KNN is useful to train a medical images dataset, that is not very large. So, the KNN algorithm characteristic of performing better with small input dataset is a strength in the medical field, that frequently faces trouble in collecting images.

- Data pre-processing: KNN is used to estimate the missing values of a dataset. This process can significantly increase the accuracy of the initial sample.

- Finance: for assessing the riskiness of a loan and the creditworthiness of a bank's client as well as stock market forecasting, as explained in the papers by Mukid *at al.* [45] and by Imandoust *at al.* [46].

- Pattern recognition: KNN is also useful for text classification and handwritten numbers identification.

- Health care: KNN can make predictions about the risk of heart attack and prostate cancer by accessing the most likely gene expressions, as explained by Bouazza *at al.* [47].

4.4 Support Vector Machine (SVM)

In 1963 the Russian statistician Vapnik proposed some algorithms for distinction and recognition learning based on finding generalized portraits of patterns with the paper "Generalized Portrait Algorithm" [48]. This inspired the researchers Boser, Guyon and Vapnik who wrote the paper "A training algorithm for optimal margin classifiers" in 1992 [49]. Thereafter, the original Support Vector Machine algorithm was published in 1995 by Vapnik *at al.* [50].

A "Support Vector Machine", or "SVM", is a powerful and versatile supervised learning algorithm that can classify cases by finding a hyperplane that best separates the features [40]. SVMs can perform both classification and regression tasks, particularly are capable to handle well linear and nonlinear classification problem.

In the case of linearly separable data, the decision boundary of a SVM classifier is a straight line (for the bi-dimensional case) or hyperplane (for the multi-dimensional case) that can easily divide the data into two groups. The point is: infinitely hyperplanes can separate the data, the one to choose should stay as far away as possible form the data points, as shown in figure 4.4.

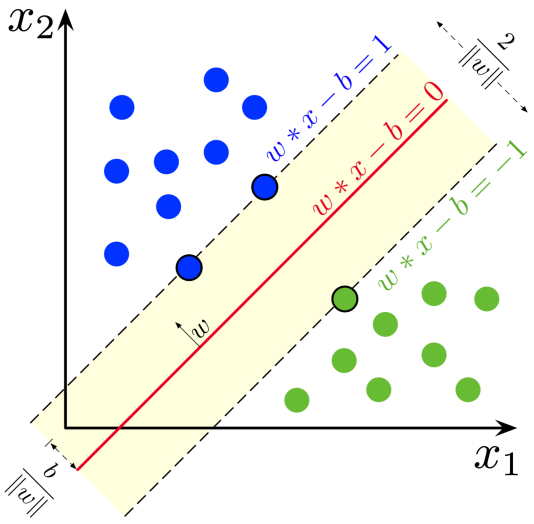


Figure 4.4: Example of bi-dimensional SVM.

This problem is called "Maximum Margin classification". The idea is to find the widest "street" possible, formed by the SVM margins (dashed lines in figure 4.4), between the two groups. The red line in the middle of the two dashed lines represents the hyperplane called "Maximum Margin classifier" and is formulated as:

$$w^T x - b = 0$$

where w is the normal vector to the hyperplane and x is a real vector of input data. The two parallel hyperplanes that define the margin are formulated as:

$$w^T x - \alpha = 0$$

$$w^T x - \beta = 0.$$

Considering for simplicity $\alpha = b + 1$ and $\beta = b - 1$, the two parallel hyperplanes became:

$$w^T x - b = +1$$
$$w^T x - b = -1.$$

So, the margins should have the largest perpendicular distance, that is calculated using the formula for the distance of a point from a plane:

$$\frac{w^T x - \beta}{||w||} - \frac{w^T x - \alpha}{||w||} = \frac{|\alpha - \beta|}{||w||} = \frac{2}{||w||}.$$

In this way the problem becomes to maximize $\frac{2}{||w||}$ or minimize $\frac{||w||}{2}$. To simplify the calculation of the gradients, is usually used the squared form:

$$\min_{w,b} \frac{1}{2}||w||^2 = \min_{w,b} \frac{1}{2} w^T w$$

with the constraint $y_i(w^T x_i + b) \geq 1$. Note that y_i can be 1 or -1 depending on the class to which the point x_i belongs. This optimization problem can be solved using the Gradient Descent method or the method of Lagrange multipliers [11] and the output will be the weights w and the bias b. In this way, a new sample will not influence the decision boundaries if added "off the street". The points that lie on the decision boundaries (circled in the figure 4.4) are called "support vectors" and variations can influence the final result. When is strictly imposed that all the points must lie "off the street", than the classification is called "Hard Margin". This type of classification works well only for linearly separable data and is sensitive to outliers.

"Soft Margin Classification" is a more flexible model that tolerates some data points to be "on the street" and, as such misclassified. The "Soft Margin" classification was proposed by the Danish computer scientist Cortes and Vapnik in 1993 and published in 1995 [50]. In this case, the maximization problem can be formulated as:

$$\min_{w,b} \frac{1}{2}||w||^2 + C \sum_{i=1}^{n} \zeta_i$$

with the constraint $y_i(w^T x_i + b) \geq 1 - \zeta_i$ for every $i = 1, ..., n$ and $\zeta_i \geq 0$. The main idea is to find a trade-off between misclassified samples and the boundaries width. This balance is controlled by the regularization parameter C. Moreover, the slack variables ζ_i make the model flexible for misclassifications. Soft Margin Classification can be a good approach also when the data are linearly separable, but the classifier is not able to find enough wide margins, so the model risks being in overfitting.

In real life, many datasets are not linearly separable. A solution can be to add polynomial features. For example, if a not linearly separable dataset is characterized by the feature x_1, adding a second feature $x_2 = x_1^2$ can result in a bi-dimensional linearly separable dataset, as shown in the image 4.5. In this way the input data space is mapped into a higher dimensional space with the transformation denoted as $\varphi(x_i)$. This method is simple to implement, but is not convenient for complex dataset. High polynomial degree creates an exponential number of new features, so

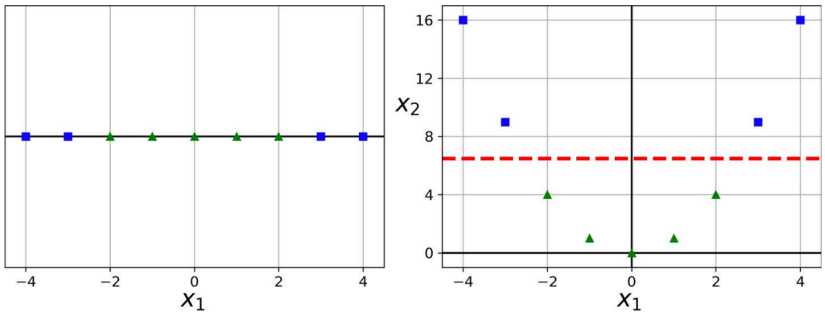


Figure 4.5: Adding features to make a dataset linearly separable. Image from book [40].

the model will face time problems. In order to face this issue, the researchers Boser, Guyon and Vapnik suggested a way to create nonlinear classifiers by applying the "kernel trick" to maximum-margin hyperplanes [49] in 1992. The "kernel trick" is a mathematical technique that handles high polynomials avoiding the calculation explosion because it will not actually add any polynomial features. The degree parameter controls the flexibility of the decision boundary. Higher degree kernels yield a more flexible decision boundary. If the model results to be in overfitting, is better to reduce the polynomial degree. Vice versa, in the case of underfitting, is better to increasing it.

In other words, the kernel maps data into a higher-dimensional space, this is called "kernelling". This trick modifies the calculation replacing the dot products by a nonlinear kernel function, denoted as $K(\vec{x_i}, \vec{x_j})$. This concept is formulated as:

$$K(\vec{x_i}, \vec{x_j}) = \varphi(\vec{x_i}) \cdot \varphi(\vec{x_j})$$

where $\varphi(x_i)$ is the function for the space transformation and "$\cdot$" denotes the dot product. In this way, the maximum-margin hyperplane can be fitted in the transformed, high-dimensional feature space. The kernel functions can be of different type, such as:

- Homogeneous Polynomial kernel: that is formulated as:

  $$K(\vec{x_i}, \vec{x_j}) = (\vec{x_i} \cdot \vec{x_j})^d.$$

  Note that, when $d = 1$, this function becomes the linear kernel.

- Inhomogeneous Polynomial kernel: that is formulated as:

  $$K(\vec{x_i}, \vec{x_j}) = (\vec{x_i} \cdot \vec{x_j} + c)^d$$

  with $c \geq 0$. c is a free parameter that controls the influence of higher-order versus lower-order terms in the polynomial. Note that, if $c = 0$, the Inhomogeneous Polynomial function is reduced to the previous Homogeneous form.

- Gaussian Radial Basis Function (RBF) kernel: tends to zero increasing the vectors distance and reaches one when the two feature vectors coincide, so when $x_i = x_j$, and is formulated as:

  $$K(\vec{x_i}, \vec{x_j}) = exp(-\gamma||\vec{x_i} - \vec{x_j}||^2)$$

for $\gamma > 0$ and where $||\vec{x_i} - \vec{x_j}||^2$ is the squared Euclidean distance between the two feature vectors. Note that $\gamma = \frac{1}{2\sigma^2}$ and σ is an hyperparameter that represents the variance.

- Sigmoid kernel function: also called Hyperbolic tangent, ranges from -1 to 1 and is formulated as:

$$K(\vec{x_i}, \vec{x_j}) = \tanh(k\vec{x_i} \cdot \vec{x_j} + c)$$

for some $k > 0$ and $c < 0$.

In order to understand which is the best function, is fundamental to compare the results of the algorithms performed using different type of kernel.

After mapping the data into a higher-dimensional space and finding the best hyperplane to divide the data into two groups, the model is ready to classify new samples. This is made plugging the new data into the hyperplane equation and calculating if the new point lies above or below the hyperplane, so if the point belongs to the first or the second category. In this way, is impossible to have a probabilistic explanation for the classification.

Originally, SVM algorithm was developed for binary classification, so it supported only the separation of the data points into two classes. Most of the classification tasks are multiclass problems, so SVM were adapted also for this type of issue. In this case, Support Vector Machines use the same original principle, breaking down the problem. The two main approaches to implement this are:

- One-to-One approach: a binary classifier is developed per each pair of classes after breaking down the problem into multiple binary classification problems. So, every two classes are separated by a hyperplane without considering the data of the other classes. So, for n classes are needed $\frac{n(n-1)}{2}$ SVMs.

- One-to-Rest approach: this method researches a hyperplane that separates a the point of a specific category from all the others. In this way, all the data are taken into account and divided into two groups: belonging or not-belonging to a certain class. This process is repeated for each category, so the number of hyperplanes corresponds to the number of classes.

SVMs are computational efficient in terms of memory, especially for small or medium-size dataset, and are accurate in high-dimensional spaces. Thanks to these characteristics, SVMs are widely used for colored image classification, as explained by Agrawal *at al.* in the paper "Content Based Color Image Classification using SVM" [16], already cited in chapter 2.1. On the other hand, SVMs underperform if the number of features is much greater than the number of samples or if the target classes are noisy.

4.5 Multiple Classifier System

Classification is not an easy task. All the technique cited in the previous chapters do not perform well for every type of dataset and application. To face this issue, some researchers developed an ensemble classifier that combine some individual classifiers, which is called "Multiple Classifier System", or MCS. This is a great new method for image classification, in particular for remote sensing images, that are images acquired by various sensors, as explained in the paper by P. Du *at al.* [51].

The main idea behind MCS is to build a set of learners, called "base learners", and aggregate them in order to end up with a model that performs significantly better than every other single model, as explained in the article by Goncalves [52]. Note that, in the case of classification, the word "learner" can be substituted with "classifier". Base learners can be algorithms as Decision Tree (see chapter 4.1), Random Forest (see chapter 4.2), Support Vector Machine (see chapter 4.4) KNN (see chapter 4.3), Neural Network (see chapter 4.6), and others. The strengths of these algorithms combined are exploited in order to produce a robust and high-quality model for pattern recognition problems, as explained in the paper by Tin Kam Ho *at al.* [53].

The process of creating a MCS can be divided in four steps: the generation of the model, the pruning of the redundant models, the selection of the best classifiers and, finally, the integration.

Firstly, a set of models is generated with one specific method, called "homogeneous" ensemble, or with more methods, called "heterogeneous" ensemble. Heterogeneous systems are more prone to achieve a good diversity than homogeneous systems. The diversity between the base learners is a fundamental concept for ensemble strategies. Find a group of very diverse learners is crucial for the realization of the overall model. In this way, a successful combination of model will provide an accurate result and the weaknesses of the models will compensate each other. Based on dependence assumptions between the learners, as explained by Fierrez *at al.* [54], the three main methods for classifier combination are:

- Bagging: is the abbreviation of "Bootstrap Aggregating" and is widely used, especially for improving the accuracy and stability of learning models such as Decision Tree or Random Forest, indeed it was already cited in chapters 4.1 and 4.2. More generally, the bagging method trains the models with random samples of the dataset, called "bootstrap samples". In this way, the algorithm is capable to increase the ensemble diversity which is fundamental for reaching a good accuracy in the results. In this way, the learners are built in parallel and, lastly, all the resulting outputs are combined with a voting or an averaging strategy (see figure 4.6).

- Boosting: uses random samples of data to train multiple learners in sequence and applies higher weights on misclassified input. In other words, the error function of a single model is influenced by the previous one. This will influence positively the prediction of the models coming next. In this way, weak learners are encouraged to a more precise classification. Finally, the results are combined with a voting or an averaging strategy (see figure 4.7). This method

results to be more accurate, but more time consuming than the previous Bagging method. This topic was already explained in the case of Decision Tree in chapter 4.1. A widely used algorithm of this type is the "AdaBoosting", that stands for "Adaptive Boosting". In this case, the data samples are selected randomly, but with a different probability of being chosen. This probability of a sample to be train in the next model depends on its misclassifications in the previous models. Another popular Boosting algorithm is the "Gradient Boosting". This is an iterative functional gradient descent algorithms that aims to built an ensemble of weak models such that the predictions of the ensemble minimize a loss function. The core idea is to use the residuals errors calculated with gradients in the loss function, instead of the weighted data [40]. An example of Gradient Boosting algorithm is the "XGBoost", or "eXtreme Gradient Boosting", that was already cited for the Tree-base Embedded methods in chapter 3.1.3.

- Stacking: first, the base learners are trained and result with intermediate predictions, that are used to create a new training set. This latter is used to train a second model, called "Meta-classifier", that ends up with the final predictions (see figure 4.8).

With the first phase many models are generated, but some of them may result to be redundant. To face this issue, a pruning is performed after the generation step in order to discard useless model and to decrease the ensemble size. In this way, both the model complexity and the amount of memory required are decreased without negatively influencing the overall accuracy.

The third phase consists in selecting the best classifiers to base the ensemble prediction on. The two main methods used are the static selection and the dynamic selection. The first one selects the classifiers during the training process. On the other hand, the dynamic algorithm performs the selection during the classification for each new test sample. So, the classifiers are chosen based on their local accuracy computed with a KNN algorithm.

Finally, the integration step involves a combination strategy to obtain the final prediction of the ensemble for new samples. The base learners can be combined with a majority voting scheme, so using the predicted class labels, or with an averaging of the posterior probabilities of each class found by the single base learners.

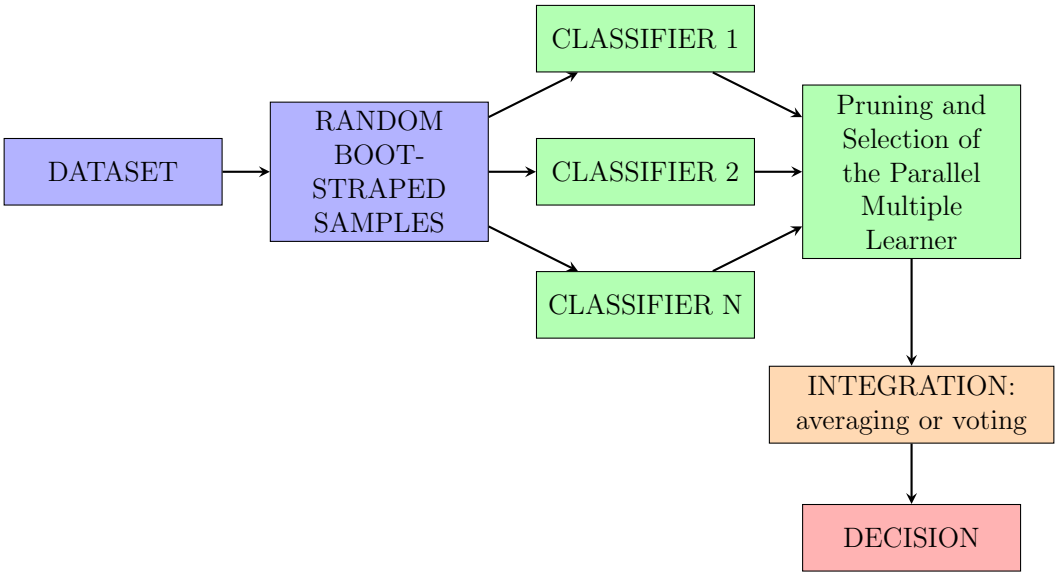


Figure 4.6: Flowchart Bagging

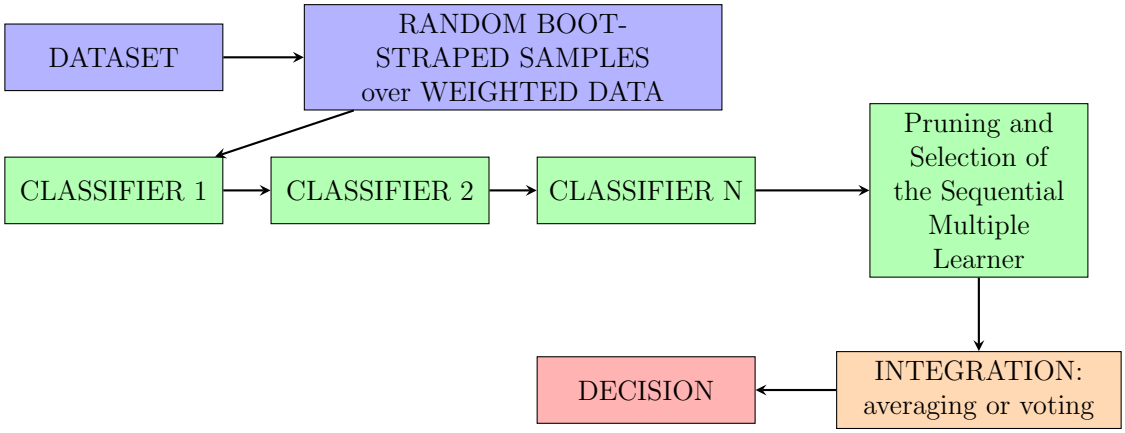


Figure 4.7: Flowchart Boosting

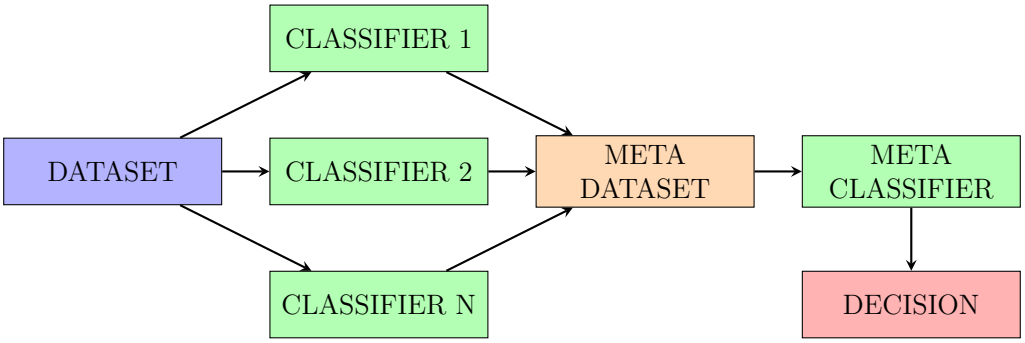


Figure 4.8: Flowchart Stacking

4.6 Convolutional Neural Network (CNN)

In more recent years, Computer Vision researchers have begun to utilize Artificial Neural Networks inspired by human vision to processes visual data and to understand how pixels relates to another. Artificial Neural Networks (ANNs) comprise mathematical models composed of artificial neurons based on the biological functioning of the human brain [55]. The human brain is composed of Neural Networks with interconnected biological neurons. "Neurons" are the basic unit of a ANN and are statistical functions able to process a huge amount of information, as the human brain does. Neurons enable each individual to reason, make in parallel processing such as recognizing sounds, images, faces or even learning and so on. Neural Networks are a valuable paradigm for solving Machine Learning engineering problems and require advanced hardware to support them.

The first idea of Artificial Neural Network was initially developed by the neurophysiologist W. McCulloch and the logician W. Pitts in 1943 [56]. In 1958 the psychologist F. Rosenblatt came up with the concept of "perceptron" [57], that is an algorithm for learning a binary classifier, in other words, is a threshold function that maps a real-valued vector as a single binary value. In 1967 the mathematician Ivakhnenko and Lapa published a paper [58] about functional networks with many layers for the first time. In those years, the researches proceeded slowly, but after the first AI winter, the American Institute of Physics established the "Neural Networks in Computing" annual meeting from 1985. However, it was only after the second AI winter that ANN reached the expectation and nowadays is still a hot topic in the Computer Vision research field.

In traditional Machine Learning, programmers design feature extractors to create learning algorithms. In contrast, Neural Networks are composed of multiple, sequential layers that are directly fed with raw data and can develop on their own the representations needed for pattern recognition. In this way, Neural Networks result to have a strong performance in the classification of objects contained in an image. Specifically, ANNs are composed of 3 types of layers, that are an assembled of neurons, as shown in figure 4.9:

- The input layer: is made of nodes, which process the input vector's values and

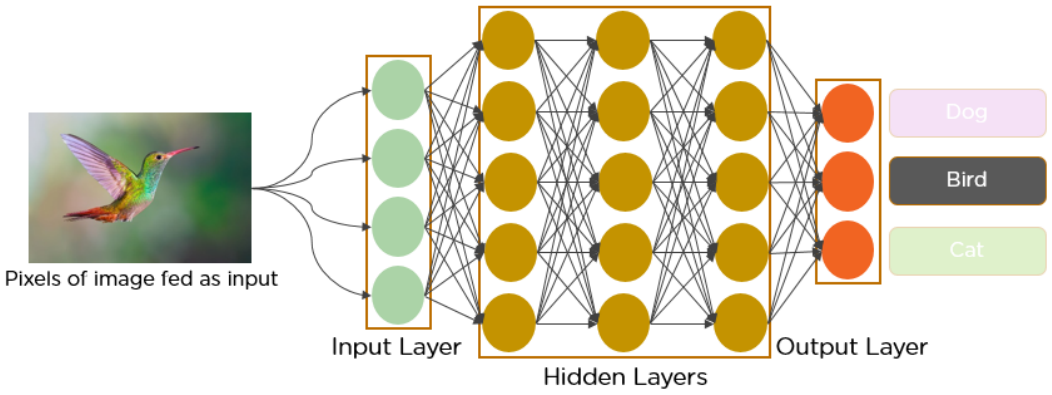


Figure 4.9: Neural Network structure.

feeds them into the dense, hidden layers.

- The hidden layers: are fully connected layers, meaning that a single node in one layer is connected to all nodes in the next layer via a series of channels. Input values are transmitted forward until they reach the output layer.
- The output layer: comprises nodes which represent the target classes and for each classes is predicted the percentage of membership.

"Convolutional Neural Network", or "CNNs", is a field of Artificial Neural Network and is an algorithm that can process and classify images using a multi-layer Artificial Neural Network. CNNs are mainly used to analyze visual imagery, in fact the first layer takes as input a 3-dimensional volume, that corresponds to a RGB tensor. The structure of a CNN is shown in detail in figure 4.10 and can be divided as:

- The input layer: its role in a CNNs is to break down the input images into an easier form to process, without losing features which is critical for achieving a good prediction. This layer has the same number of neurons as number of pixels in the image for each of the RGB channels.
- The Convolutional layers: are usually made of 32 weighted matrices, called Filters or Kernels, which are defined by their width, height, and depth. The concept of Image Filtering was already explained in chapter 1.2. Convolutional layers apply filters to the original image in order to extract high-level features. The first convolutional layer captures the shape of the edges, the colors, the gradient orientation, and so on. This type of features is called "low-level features". Moving forward with additional layers, increasingly higher-level features are found. In this way, the model has a complete understanding of the input images. One of the main disadvantages of CNN is the interpretability of the feature extracted. Many feature, especially high-level features, are not interpretable at all, as explained by Molnar *at al.* in his book [59]. Moreover, only some low-level feature can be visualized in the training process. The convolutional layers result in a 3-dimensional tensor, composed of a 2-dimensional map per filter. The latter is fed into an elementwise activation function that determines if a node will "fire" or not given the input data. In other words, the signal from the previous cell is processed and converted in a suitable form for the next layer, discarding the irrelevant information. An activation function is a non-linear transformation. The most used activation function are the "ReLU", the "Sigmoid" (see equation 4.2), the "SoftMax" (see equation 4.4) and the hyperbolic tangent (see equation 4.3).

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{4.2}$$

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{4.3}$$

In the case of a Convolutional layer, the "ReLU", or "Rectified Linear Unit", function is the most widely used because is the only one that convert to zero

negative input and is formulated as $max(0, x)$. In this way, the neurons are not activated all at the same time, so the model results to be sparse and more efficient.

- The Pooling layer: is a down-sampling operation for bidimensional spatial data. It is important for decreasing the dimensions of the data propagating through the network. This is made extracting the maximum or the average of an area using a sliding window approach. Moreover, Pooling is crucial for object recognition tasks because is able to provide the spatial variance of an image.

- Convolutional and Pooling layers are repeated to extract higher-level features.

- The Dense layer is a "fully connected" layer. This means that all the neurons in a layer are connected to those in the next layer. In the Dense layer the information is aggregated and the results are flattened before the classification in N groups, so the output is a N-dimensional vector. This is made performing matrix-vector multiplication and the matrix parameters are updated with backpropagation. An important issue is that the backpropagation requires a huge amount of RAM, especially during the training, because all the intermediate values computed during the forward pass are stored [40]. Finally, the "SoftMax" activation function is used to convert the output of the neuron into the membership probability of each attribute. The "SoftMax" function is a type of Sigmoid function that can handle classification tasks ending up with the probabilities of membership for each class. It is formulated as:

$$\frac{e^x}{\sum_j e^{x_j}}. \tag{4.4}$$

- The Output layer: is a Dense layer with a number of neurons that corresponds to the number of target classes and is composed of the probability of belonging to each category.

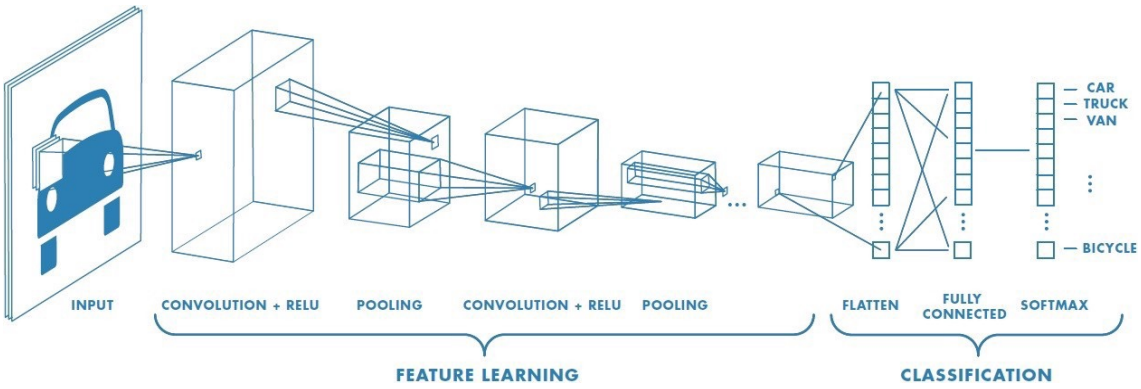


Figure 4.10: Convolutional Neural Network structure.

CNNs are difficult to apply to high resolution images, but to face this problem Krizhevsky *at al.* developed a CNN called "AlexNet" for large scale visual

recognition in 2012 [60] studying GPUs optimization. The starting dataset for this challenge is called "ImageNet" and contains about 15 million high-resolution images belonging to 22000 classes. AlexNet architecture is composed of eight layers, including convolutional layers, max-pooling layers and fully connected layers characterized by a "ReLU" activation function, that results in a better performance than other functions. This network became popular because it achieved a very good accuracy with an error of only $15,3\%$. Researchers explained that using GPUs was crucial for high performance with feasible computational costs.

To sum up, the core idea of a NN is tuning the parameters in order to map the input, an image in the case of a CNN, to an output, such as a label. The number of parameters is usually in the order of millions, so a large amount of data is required as input to achieve a good performance. In real world scenarios, the lack of data available is a crucial issue for Deep Learning algorithms. To face this problem, Image Augmentation is often performed, as explained in chapter 1.1. Thanks to this technique, minor alterations, such as flipping, translations, cropping or rotations, are performed on existing data. In this way, the amount of relevant data is increased and the CNN results to be invariant to translation, size and illumination and, as such, more robust.

An advantage is that CNNs do not need pre-processing techniques, except to Image Augmentation. CNNs are able to adapt to images during the training process without human intervention. On the other hand, often CNNs are used as supervised learning algorithm, meaning that images are labeled before the training. The labeling process is called "image annotation" and usually requires the human intervention. In this way, the class of membership is known for every input image. This is fundamental because enables the model to evaluate errors in the classification.

Convolutional Neural Networks are widely diffused in Computer Vision, especially in the medical field. For instance, CNNs are used for the segmentation of anatomical tissues images, such as for the prevention of brain tumors, as explained in the book [11]. Moreover, CNNs can be applied not only to simple images, but also for video processing and volumetric image processing.

4.7 Transformers

In 2018 Google researchers built a new architecture called "Transformer" that exploits the so called "attention mechanisms" to increase Deep Neural Networks efficiency. This concept was firstly introduced by Vaswani *at al.* [61] in 2017. In first place, the new architecture is used specially for Natural Language Processing (NLP) tasks, such as translation, then was extended to Computer Vision tasks, outperforming CNNs.

For the translation of a sentences, a model needs to figure out dependencies and connections between words in previous sentences. For this type of problem, Recurrent Neural Networks (RNNs) are often used. In RNNs each word is processed separately and information persist to ensure an appropriate translation. RNNs assume that all the input words are equally important, so a problem arises when the sentences are very long. In this case, RNNs performance decreases significantly because the architecture is not always able to remember all the long-term dependencies between words and some of them risk being discarded. For example, if the input text is too long, the model can face difficulties in finding relations between the words in the first and in the last sentences. In other words, the algorithm is not able to prioritize input and to selectively remember or forget information considered more or less important. In addition to this, RNNs process words sequentially, and this characteristic inhibits parallelization increasing the time consumed for the task.

In order to face these issues, a "Self-Attention" mechanism is developed to enable Neural Networks focusing on the important parts of the input. This algorithm tries to capture the relations between the words in the sentences. It calculates the relative importance of the inputs, called keys, for a particular output, called query, using dynamic weights that are multiplied by the input sequence, called values. In practice, if the sentence to translate from English to Italian is composed of the queries:

"Anna" "drinks" "a coffee" "every morning" "because" "it" "helps" "her" "weaking up."

the first part analyzed is the first word, so the subject *"Anna"*. For the human brain is natural thinking about *"drinks"* as the *"verb"* referred to the subject *"Anna"* and translating it in second place. For the model is not trivial to figure out what is the verb that refers to the subject *"Anna"*. Moreover, the model must associate *"Anna"* with *"her"* and *"a coffe"* with *"it"*. To face this problem, a similarity measure, such as the dot product [40], is computed between the query and each key. In this way, it is determined the importance of each relations between words. Then, the similarity scores are converted in weights that ranges from 0 to 1. If the weight of the words *"a coffe"* and *"it"* tends to 1, means that the two words are highly correlated and refer to the same thing.

In other words, an attention function maps a query q and a set of key-value pairs (k, v) of dimensionality d_{key} computing the weighted sum of the values [61]:

$$\sum_i w_i v_i \tag{4.5}$$

where w_i is non-negative and $\sum_i w_i = 1$. The weights are obtained by taking the Softmax function, explained in chapter 4.6, of the dot product between the query

and the key vector in order to scale the weights in the range $[0, 1]$:

$$w_i = Softmax(q_i \cdot k_j). \tag{4.6}$$

In order to increase the computational efficiency, the attention function on a set of queries can be computed simultaneously. To do this, the queries, the keys and the values are packed respectively in the matrices Q, K and V. This technique is called "Scaled Dot-Product Attention" and in this case the equation 4.6 becomes:

$$W = Softmax\left(\frac{Q \cdot K^T}{\sqrt{d_{keys}}}\right) \tag{4.7}$$

where Q is a matrix containing one row per query, K is a matrix containing one row per key and V is a matrix containing one row per value. Moreover, the scaling factor $\frac{1}{\sqrt{d_{keys}}}$ downscales the similarity scores to ensure not too large values produced by the dot product for the Softmax function and avoid the gradient vanishing problem. Similarly to equation 4.5, the final output matrix results to be:

$$Attention(Q, K, V) = W \cdot V = Softmax\left(\frac{Q \cdot K^T}{\sqrt{d_{keys}}}\right) \cdot V \tag{4.8}$$

The "Multi-Head Attention" approach linearly projects keys, values and queries in order to perform the attention function in parallel on each of the projected queries, keys and values. The outputs are concatenated and projected back to the original space resulting in the final values. So, a "Multi-Head Attention layer", represented in orange in figure 4.11, is composed of several Scaled Dot-Product Attention layers that apply multiple different linear transformations to find many different projections. In this way, the model learns information from different representation subspaces at different positions and many different characteristics of the word are found. The model can understand for each word not only the proper key, but also the position in the text or if a word is repeated multiple time in the past sentences.

The architecture of a Transformer, shown in figure 4.11, can be divided in four parts:

- Input Embedding: the text input is transformed into a format understandable by the Transformer. First, the input is broken down in chunks, called "tokens", that can be processed separately. Each token is represented by a list of numbers and are then converted into vectors, called "word embeddings". These vectors lie on a a multi-dimensional space, called "embedding space", that aims to capture the meaning of the words, such that words with similar meaning are closer together. To do this, usually are used embedding models for language tasks that are pre-trained separately.

- Positional Encoding: if several sentences are fed into the model, each word may have a different meaning in respect to which sentence belongs. The Multi-Head Attention layers has access to each word position in the sentence thanks the "Positional Encoding". This is a dense vector PE and is calculated with

oscillating functions, as shown in equations 4.9 and 4.10, in order to end up not only with the absolute positions, but also with the relative positions. In his paper [61], Vasvani explains that sine and cosine functions allow the model to learn also relative positions because any PE_{pos+k} can be represented as a linear function of PE_{pos}.

$$PE_{pos,2i} = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right) \tag{4.9}$$

$$PE_{pos,2i+1} = \cos\left(\frac{pos}{10000^{2i/d_{model}}}\right) \tag{4.10}$$

For every even time step, the equation 4.9 is used; similarly, equation 4.10 is applied for each odd time step, then the final vectors are concatenated. Moreover, in equations 4.9 and 4.10 *pos* is the position in the sentence and i is the dimension.

- The encoder: is the lefthand part of figure 4.11 and takes as input some sentences and each word position in the sentence, thanks to the Positional Encoding. It is composed of $N = 6$ identical layers, as explained in the paper [61], composed of two sub-layers: a multi-head self-attention mechanism, colored in orange, and a position wise fully connected feed-forward network, in blue. As explained before, the Multi-Head Attention layer encodes each word's relationship with every other word in the same sentence, focusing more on the most important ones. Moreover, each of the two sub-layers are followed by a normalization layer, so each sub-layers end up with a 512-dimensional output. Note that, in the encoder, Q represents the current position-word vector in the input sequence and K and V represent all the position-word vectors in the input sequence.

- The decoder: is the righthand part of figure 4.11, and takes as input not only the target sentence shifted one time step to the right, but also the output of the encoder for N times. Similarly to the encoder, the decoder is composed of identical layers repeated $N = 6$ times. The decoder has three sub-layers: a Masked Multi-Head Attention layer for the input sentence, a Multi-Head Attention layer for the output of the encoder and a normalization layer. The first sub-layer is "Masked" in order to consider only the words before the current one. This ensures that the final prediction for a certain position will not depend on subsequent positions. In this way, at every time-step the model predicts one more word, which will be taken as input by the decoder at the next time-step [40].

Nowadays, a widely researched application of Transformers is for Computer Vision tasks, outperforming CNNs. The convolutional operation assumes that neighbor pixels are more important than distant pixels and this assumption can lead to misleading results, but this issue can be solved by the Self-Attention mechanism [11]. In the paper "The Image Transformer", Vaswani *at al.* [62] explains the idea of applying an Attention mechanism to CNNs. So, a Transformer is used on images in order to predict the next pixel, given a sequence of input pixels and the predicted

ones. On the other hand, Transformers have difficulties dealing with large-size images as input. In this case, the number of operations required for the Self-Attention mechanism is significantly larger than using convolution operation. In 2020, Dosovitskiy, Beyer *et al.* [63] argued that transformers can process large images with a model called the "Vision Transformer", or "ViT" for image recognition task. Each pixels is no more treated as a single input, but the image is divided in gridded image patches and each of them is flattened and combined with a positional encoder. In this way, the Attention mechanism is deployed on multiple image patches rather than the entire image.

In recent years, the Vision Transformers have been applied also for image classification tasks, as explained in the article written in 2021 by Bazi *at al.* [64]. Bazi explains how the idea of Dosovitskiy *et al.* [63] can be used as a remote-sensing scene-classification method. Firstly, the images are resized to 224×224 and the patch size is fixed to 16×16, so the resulting patches are converted to 196-length tokens by flattening and embedding. Then, the Positional Encoding is added to the patches to keep the position information. The result is fed to several Multi-Head Attention layers for generating the final representation. Finally, a Softmax classification layer is used for classification. Moreover, the classification performance is increased because of data augmentation strategies to generate additional data for training.

To conclude, Self-Attention layers result to be more efficient compared with the recurrent and convolutional layers for the computational complexity [61]. Furthermore, these layers allow to parallelize more computation minimizing the number of sequential operations and decreasing the time required. Unlike RNN models, Transformers are more efficient because they do not take one word at a time as input, but can receive an entire embedded sentence in parallel. In this way, Transformers can examine the context of the text in both forward and backward sequences. Finally, it is important to mention that Transformers can learn long-range dependencies, without the need of huge labeled dataset as input, by using to the ability of finding patterns mathematically.

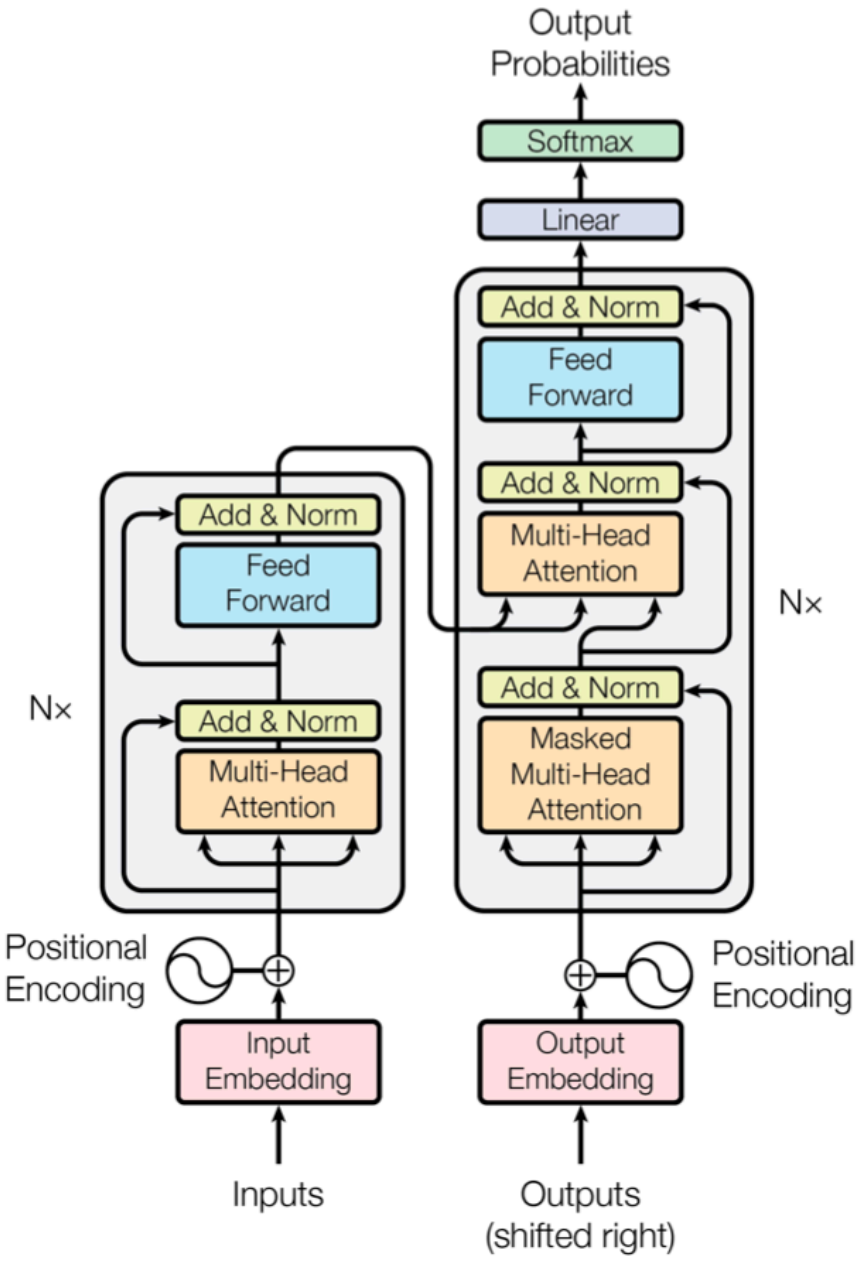


Figure 4.11: The Transformer: model architecture. Image from paper [61].

Chapter 5

An Application Study

This chapter describes my internship project, which was developed in the Inspection Machine department of Antares Vision S.p.A [65]. This company produces software for pharmaceutical products inspection machines. The main subject of the project is the creation of an image classification process for closure caps of pharmaceutical ampoules. These images are acquired by Antares Vision inspection machine and are in "png" format.

The task is solved by the development of algorithms based on Python programming language, in particular using the OpenCV library. The project deals with various topics which are analyzed in detail in the previous chapters.

5.1 The Experiment

This project's main scope is to classify Flip-off's images. Flip-offs are plastic closure caps of pharmaceutical ampoules. The classification is based on the "Target", that ranges from 0 to 8. Every Target corresponds to a type of flip-off, that are: "Black-Silver Transparent", "Black-Gold Transparent", "Red-Silver Transparent", "Red-Gold Transparent", "Alluminium", "Orange with text", "Orange" and "Blue", as shown in figure 5.1.

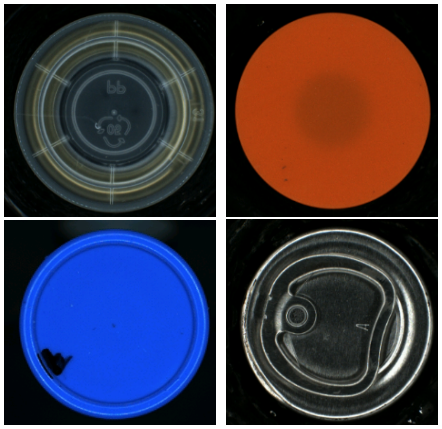


Figure 5.1: Examples of "Black-Gold Transparent", "Orange", "Blue" and "Alluminium" Flip-offs.

First, several algorithms are applied to the image in order to segment the contours of the caps called "Edge Detection", see chapter 1.3, and then the "Hough Transformation", see chapter 1.4, to mark the cap contour circle. Considering only the image within the circle defined above, the following characteristics are extracted:

- Circle area;
- RGB levels*: mean, variance and standard deviation of red, green, blue, see chapter 2.1;
- HOG*: mean, standard deviation, minimum and maximum;
- SIFT*: mean and standard deviation.

*The marked features are calculated considering the inner area of the circle (80% of the circle radius found), so that background or unnecessary shadows are not considered.

As explained in the previous chapters 2.2 and 2.3, "HOG" means "Histogram of Oriented Gradients", a technique that counts the occurrences of gradient orientation in localised portions of an image to describe its shape, and "SIFT" stands for "Scale Invariant Feature Transform", an algorithm that extracts the key points of an image, i.e. those points that remain unchanged by modifying the image with rotations, zooms, and so on.

The classification problem was divided into two macro-problems, as shown in figure 5.2. In the first, transparent flip-offs are grouped together and the classification is reduced to: transparent, orange, orange with text, blue and aluminium with a ring. The second focuses on the classification of the various types of transparent flip-offs, which can be: black-silver, black-gold, red-silver and red-gold. In both cases, the datasets were randomly divided into training set (70%) and validation set (30%).

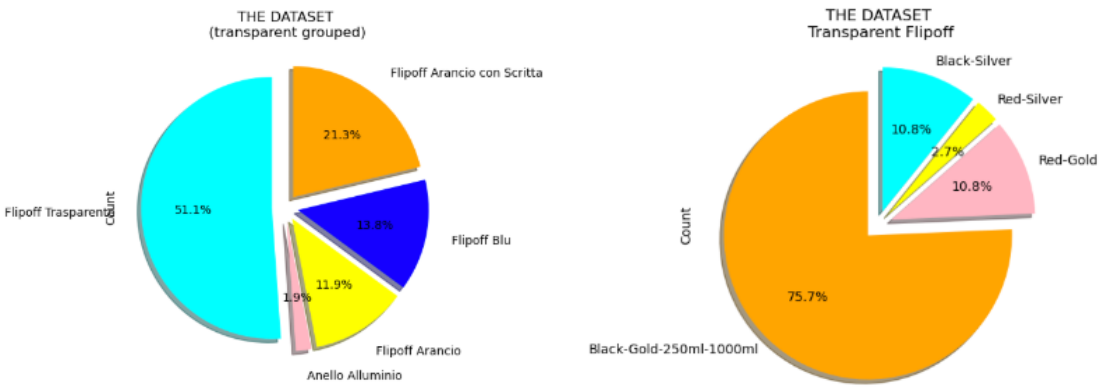


Figure 5.2: The two macro-problems.

For both problems, embedded feature selection algorithms, such as Lasso regression and tree-based selection, are performed to decide which are the best features to use for classification, see chapter 3.1.3. Finally, the folllowing Machine Learning algorithms have been applied: Linear and 3rd degree Support Vector Machine, Decision Tree, Random Forest and K-Nearest Neighbors, see respectively chapters 4.4,

4.1, 4.2 and 4.3. In both problems, the classification of these algorithms turns out to be accurate, in particular Random Forest and Decision Tree have an accuracy ranging from 97% to 100%. Moreover, a Graphical User Interface (GUI) has been implemented, which shows the whole project interactively.

Finally, a Convolutional Neural Network, see chapter 4.6, and a Vision Transformer, see chapter 4.7, were implemented in order to classify all the images, without a subdivision into macro-problems. The CNN is a type of Artificial Neural Network in which the pattern of connectivity between neurons is inspired by the organisation of the animal visual cortex. The Vision Transformer is also based on Artificial Neural Networks, but divides the images into patches and uses the Positional Encoding and an "Attention Mechanisms" to build several Multi-Head Attention layers. Finally, the Transformer classifies the images with a Softmax classification layer. Since both the CNN and the Transformer take as input only the images, and not the extracted features, and the number of images collected is not large enough to guarantee a successful classification with this type of algorithm, a "Data Augmentation" is performed. As explained in chapter 1.1, this technique artificially expand the number of images in the dataset by creating modified versions of images, for example zooming, cropping or rotating the image. Note that the flipping is not considered as an augmentation method because it leads to the creation of unreal images in the case of flip-offs with a text. For both these models, the accuracy of the network can be set by the user directly from the GUI.

Considering that the images are resized 250 x 250, the CNN results are:

Accuracy:	Number of epochs:	Time:
30%	4	35 sec.
50%	14	2 min.
60%	17	2 min. and 25 sec.
70%	13	2 min.
80%	16	2 min. and 20 sec.
90%	20	3 min.

Table 5.1: CNN results.

The Vision Transformer takes as input images reshaped 224 x 224 and diveded in 16 x 16 patches, as suggested by the paper [63], and the results are:

Accuracy:	Number of epochs:	Time:
30%	2	18 sec.
50%	2	18 sec.
60%	9	1 min.
70%	70	8 min.

Table 5.2: Vision Transformer results.

The Vision Transformer does not exceed the 70% of accuracy, but reaches a plateau at 71% after about 70 epochs. Unlike in this case, consider that the results

reported in the paper [63] are achieved by pre-training on a large amounts of data the Vision Transformer model.

Note that, in both the models, the inserted accuracy is achieved for both the training set and the validation set in order to stop the epochs. Moreover, since the creation of the training an the validation sets is randomized, these results can vary and the time needed for the training depends also on the characteristics of the machine used.

To sum up, the whole project flowchart is clearly represented in figure 5.3.

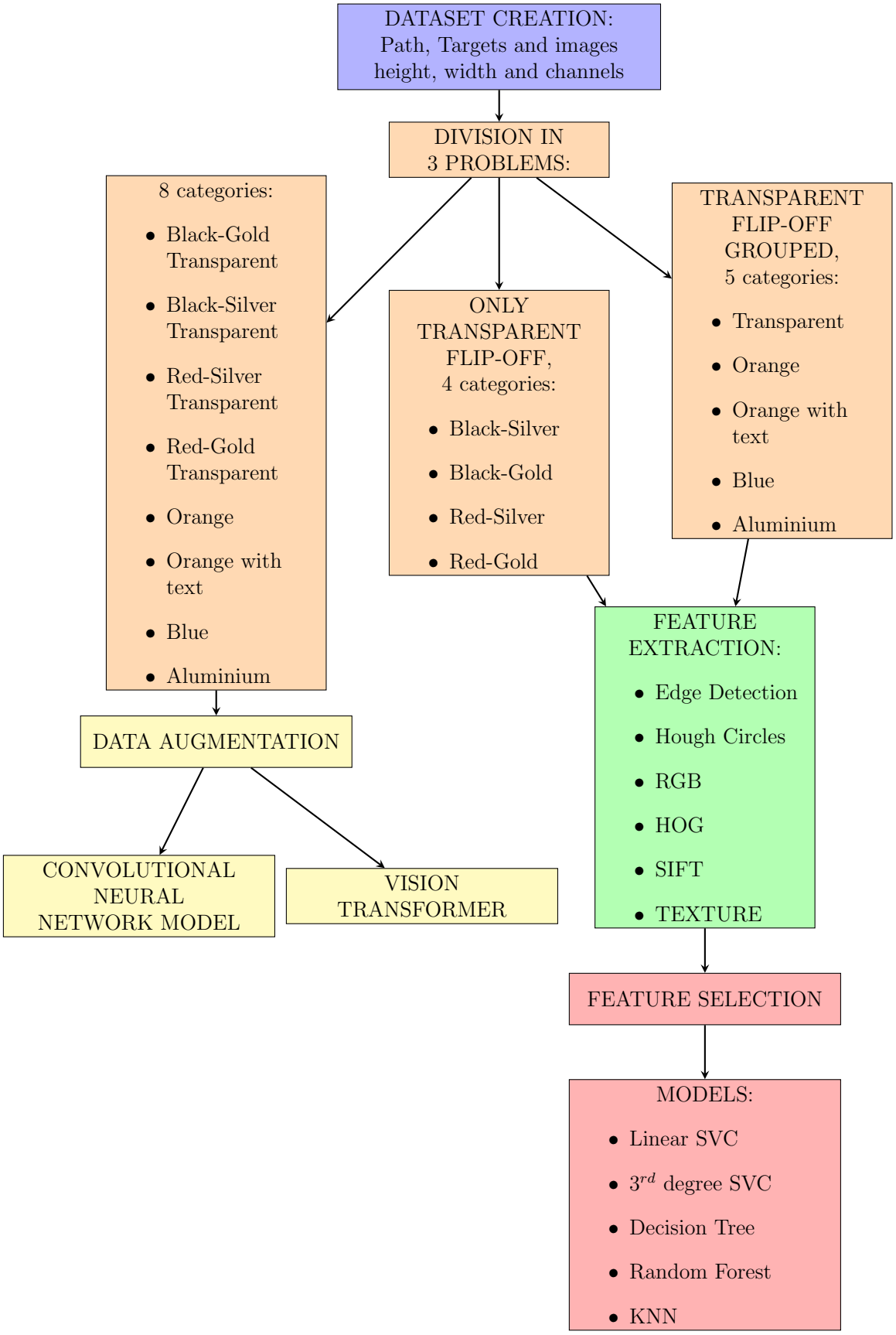


Figure 5.3: Project Flowchart

5.2 Technical Analysis

Below, a brief summary of the various Python files is presented to implement all the algorithms previously explained. The files are numbered and should be runned in the given sequence to achieve the expected result. All the files can be found on GitHub at the link [66].

- *1_1_notonlygood_dataset_creation.py*: creation of the dataset called "*df_ notonly_AllGood.csv*" with images Path, Targets and images height, width and channels. This file corresponds to the first blue rectangle in the flowchart 5.3.

  Firstly, the path to the folder called "*AllGood*" is needed, that is a directory composed of nine sub-directories, each with the images of a specific type of flip-off in "png" format. Each of these images is read with the function "*cv.imread()*" in order to extract the height, the width and the number of channels of every images. In this way, a dataframe called "*dfAllGood*" is created and saved as "*df_notonly_AllGood.csv*" with the information previously extracted in addition to a column for the path for every image and a column for the target classes numerated from 0 to 8:

  0. Black-Gold Transparent 250 ml, (134 images);
  1. Black-Gold Transparent 1000 ml, (6 images);
  2. Red-Gold Transparent, (20 images);
  3. Red-Silver Transparent, (5 images);
  4. Black-Silver Transparent, (20 images);
  5. Aluminium, (7 images);
  6. Orange, (43 images);
  7. Blue, (50 images);
  8. Orange with text, (77 images);

  for a total of 362 images. In this file the Python libraries mainly used are: "*pandas*", "*numpy*", "*cv2*", and "*os*".

- *2_1_notonlygood_edge_and_color_circle_HOG.py*: this file implements the Feature extraction step, which corresponds to the second green rectangle in the flowchart 5.3.

  Firstly, the path to the folder called "*stage_project_py*" is needed in order to have access to the csv file created previously. Then, the images contrast is adjusted using the function:

  convertScaleAbs(image, alpha, beta)

  where alpha controls the contrast and ranges from 1.0 to 3.0, and beta controls the brightness and ranges from 0 to 100. In this way, the Canny Edge Detection (see chapter 1.3) is easier to perform on each image with the function:

cv.Canny(adjusted image, high threshold, low threshold)

where a high value of intensity gradient threshold dictates that any contrast above its value will be immediately classified as an edge. The resulting images with just white edges and black background are saved in the "EdgeDetection" folder and divided in 9 sub-directuries numbered from 0 to 8, in respect to each belonging classes. The Hough Circles algorithm (see chapter 1.4) is performed on each image with the function:

cv.HoughCircles(edge_img, cv.HOUGH_GRADIENT, dp=1, minDist, param1, param2, minRadius, maxRadius).

This function takes as input:

- the Edge Detection images;
- the method used called "HOUGH _GRADIENT";
- $dp = 1$ to have the same resolution of the input also for the output image;
- the minimum distance between the centers of the detected circles;
- *param1* is the gradient value used in the edge detection;
- *param2* is the accumulator threshold for the circle centers (the smaller it is, the more false circles may be detected);
- the minimum and maximum radius length of the circle to find (see link to the OpenCV documentation [67]).

The resulting images with a green circle that highlights the cap outline are saved in the "*HoughCircles*" folder and divided in 9 sub-directuries numbered from 0 to 8, in respect to each belonging classes. By using these algorithms, to the dataset "*df_notonly_AllGood.csv*" are added the columns "CircleArea", "x_circle", "y_circle" and "r_circle" that describe the characteristics of the resulting circles.

These information are used for masking the 80% of the radius in order to consider only the internal part of the cap colouring in black the external part. In this way, the mean, standard deviation and variance of the RGB levels (see chapter 2.1) are extracted without considering the inner area of the circle, so that background or unnecessary shadows are not taken into consideration. In the following step, the following columns are added to the dataset: "r_mean_circle", "g_mean_circle", "b_mean_circle", "r_var_circle", "g_var_circle", "b_var_circle", "r_std_circle", "g_std_circle" and "b_std_circle". Note that the next algorithms will be applied on the masked image, in order to focus the attention only on the center of the closure caps.

The Histogram of Oriented Gradients algorithm (see chapter 2.2) is performed by the "*skimage*" library with the function:

hog(img, orientations=9, pixels_per_cell=(16, 16), feature_vector=True, cells_per_block=(2, 2), block_norm='L2', visualize=True, multichannel=True)

that takes as input:

- an image;
- the number of orientation bins;
- the size in pixels of a cell;
- if the output will include also a feature vector by calling the function "*.ravel()*";
- the number of cells in each block;
- the block normalization method;
- if the output will return also an image of the HOG;
- if this image will be colored or not.

The output of this function comprehends both the list of features and the images with the oriented gradients and black background, that are saved in the "HOG" folder and divided in 9 sub-directuries numbered from 0 to 8, in respect to each belonging classes. The lists of feature, the minimum, the maximum, the mean and the standard deviation of these lists are added to the dataset.

The Scale-Invariant Feature Transform algorithm (see chapter 2.3) is performed by the "*OpenCV*" library with on each image by using the function:

$$kp = cv.SIFT\_create().detect(gray\_img,\ None)$$

that takes as input a gray-scale image to detect the key points. The lists of these points, the mean and the standard deviation of these lists are added in the dataset. Moreover, the key points are drawn on the images using the function:

$$cv.drawKeypoints(gray\_img,\ kp,\ img,\ flags = cv.DRAW\_MATCHES\_FLAGS\_DRAW\_RICH\_KEYPOINTS)$$

that takes as input:

- a gray-scale image;
- the extracted key points;
- the colored image;
- a flag that indicates if the output will show the key-points with colors.

The resulting images with the key point marked in various colors and black background are saved in the "SIFT" folder and divided in 9 sub-directuries numbered from 0 to 8, in respect to each belonging classes.

Finally, Haralick Texture algorithm (see chapter 2.4) is performed on the masked part of each image. The function used is from the "*mahotas*" library:

mahotas.features.haralick(gray_img)

that takes as input a gray scale image and calculates Haralick texture features. Moreover, also the mean of these texture features will be added in the dataset.

By deploying these algorithms, to the dataset are added the columns called: "HOG", "HOG_max", "HOG_min", "HOG_mean", "HOG_std", "SIFT", "SIFT _mean", "SIFT_std" and "TEXTURE".

- *3_models_transparent_grouped.py*: as in the other files, the path to the folder called "*stage_project_py*" is needed first in order to have access to the csv file previously created. Than, transparent flip-off are grouped in a unique class called "Transparent" and a new columns are added to the dataset called "New Target" and "New Target Name" with respectively the number and the name of the new classes. This file corresponds to the right-hand orange and red rectangles in the flowchart 5.3. The features saved in the dataset are selected with two different Embedded methods: a regression algorithm, called "LogisticRegression" using the library "*sklearn.linear_model*", and a tree-based method, called "XGBClassifier" using the library "*xgboost*" (see chapter 3.1.3). The feature chose in respect to the selection are: "r_var_circle", "b_var_circle", "g_var_circle", "HOG_std', "CircleArea" and "SIFT_mean". At this point, the images are randomly splitted in training (70%) and validation set (30%) and classified using some the chosen features with the algorithms: Linear and 3° degree SVM, KNN with $K = 3$, Random Forest Classifier with 5 trees and Decision Tree Classifier (see chapter 4) using respectivelly the functions from "*sklearn*" library:

  LinearSVC().fit(X_train, y_train)

  svm.SVC(kernel='poly', degree=3, C=1).fit(X_train, y_train)

  KNeighborsClassifier(n_neighbors=3).fit(X_train, y_train)

  RandomForestClassifier(n_estimators=5).fit(X_train, y_train)

  DecisionTreeClassifier().fit(X_train, y_train).

  Finally, a new dataframe called "*test_models/model_accuracy.csv*" is created to save the accuracy and the F1 score of each model. Moreover, for each model the flip-off images of the validation set with written the ground truth and the prediction of each model are saved in the folder called "*test_models*".

- *4_transparent.py*: corresponds to the centered orange and red rectangles in the flowchart 5.3. So, only the transparent flip-offs are considered. Moreover, the class "Black-Gold Transparent 250 ml" and "Black-Gold Transparent 1000 ml" are united in an unique category called "Black-Gold Transparent".

- *5_models_transparent.py*: transparent flip-off, that were grouped in file "*4_transparent.py*", are classified using the same models cited in file "*3_models _transparent_grouped.py*". The feature used for the classification are: "r_mean

_circle", "g_mean_circle", "b_std_circle", "SIFT_std", "HOG_ min" and "r_var _circle". These are selected with the same Embedded methods of the previous case. Finally, a new dataframe called "*test_models_trasparent/model _accuracy.csv*" is created to save the accuracy and the F1 score of each model. Moreover, for each model, the flip-off images of the validation set with written the ground truth and the prediction of each model are saved in the folder called "*test_models_trasparent*".

- *GUI_stage.py*: a Convolutional Neural Network (see chapter 4.6) and a a Vision Transformer (see chapter 4.7) are performed using mainly "*keras*" and "*tensorflow*" libraries for the classification of the whole dataset, that corresponds to the left-hand orange and yellow rectangles in the flowchart 5.3.

  The images are first masked and cropped, in order to feed these models only with the inner part of the closure caps images. Moreover, the images are randomly divided in in training (70%) and validation set (30%). Than, a Data Augmentation (see chapter 1.1) is performed using the function from the "*tensorflow.keras.preprocessing.image*" library:

  ImageDataGenerator(zoom_range= 0.2, rotation_range= 10, fill_mode= 'nearest')

  that takes as input the range for random zooms, the degree range for random rotations and how the points outside the boundaries of the input are filled.

  The CNN is built with four layers by considering the "*tensorflow.keras.layers*" and each of them comprehend:

  - Convolutional layer: with a ReLu activation function, that is:

    Conv2D(32, (3, 3), activation='relu').

  - Last layer: instead of the previous convolutional layer, is a fully connected layer with a ReLu activation function, that is:

    Dense(512, activation='relu').

  - Batch Normalization layer: that optimizes the model performance applying a transformation that maintains the mean output close to 0 and the output standard deviation close to 1. The layer used is:

    BatchNormalization().

  - Pooling layer: for decreasing the dimensions of the data propagating through the network, that is:

    MaxPooling2D(pool_size=(3, 3)).

  - Drop-out layer: that discards the 40% of the data in order to avoid the overfitting:

    Dropout(0.4).

Finally, the output layer has a SoftMax activation function and 9 neurons that correspond to the 9 classes to predict, that is:

Dense(9, activation= 'softmax').

The Vision Transformer interprets an image as a sequence of patches and processes it by deploying a standard Transformer encoder. This is widely used in Natural Language Processes, as suggested by the paper [63]. Firstly, several layers are implemented, then these layers are build together to come up with the model architecture. Secondly, the Patch Creation layer takes a 224x224 image and transforms it into a grid of 16x16 patches using the function:

tf.image.extract_patches(images=images, sizes=[1, self.patch_size, self.patch_size, 1], strides=[1, self.patch_size, self.patch_size, 1], rates=[1, 1, 1, 1], padding="VALID",)

that takes as input:

- the images;
- the images shape;
- the stride length: describes how far the centers of two consecutive patches are in the images, so if these patches will be overlapping or not;
- rates: specifies how far two consecutive patch samples are in the input;
- padding: is set to "VALID", if every patch is fully contained in the image, or to "SAME", if the patches are allowed to be incomplete and the remaining pixels will be filled in with zeroes.

The Patch Encoder layer will linearly transform a patch by projecting it into a vector of size *projection_dim* using a Dense layer. Than, a learnable position embedding is added to the projected vector using the *"keras"* function:

layers.Embedding(input_dim = num_patches, output_dim=projection_dim)

that takes an index code into a position in an embedding table and returns a vector.

Finally, the Vision Transformer model consists of the Patch Creation layer and the Patch Encoder layer (previously implemented), and several repeated layers:

- Multi-Head Attention layers: to apply the Attention mechanism to the sequence of patches:

  layers.MultiHeadAttention(num_heads=num_heads, key_dim=projection_dim, dropout=0.1).

- Normalization layers: to optimize the model performance:

$$layers.LayerNormalization(epsilon{=}1e-6)(encoded\_patches).$$

- Flatten layers: to reshape the image representation:

$$layers.Flatten()(representation).$$

- Drop-out layers: to discard the 50% of the data in order to avoid overfitting:

$$layers.Dropout(0.5).$$

- Multi-layer perceptrons: that is composed of several dense and drop-out layers:

$$mlp(representation,\ hidden\_units{=}\ mlp\_head\_units,\ dropout\_rate{=}0.5).$$

- Dense layer: to produce the final class probabilities output with a Softmax activation function:

$$layers.Dense(num\_classes)(features).$$

For both the CNN and Vision Transformer, the epochs will stop when the model achieves the accuracy insert from the user. When the number of epochs reaches 100, the training will stop even if the inserted accuracy is not achieved yet. The inserted accuracy is validated and an error message is displayed if the character entered is different from a number that ranges from 0 to 1. Moreover, the graphics of the training set and validation set accuracies and losses are generated. In addition, the flip-off images of the validation set with written the ground truth, the prediction of the models and the time needed for the training are visualized.

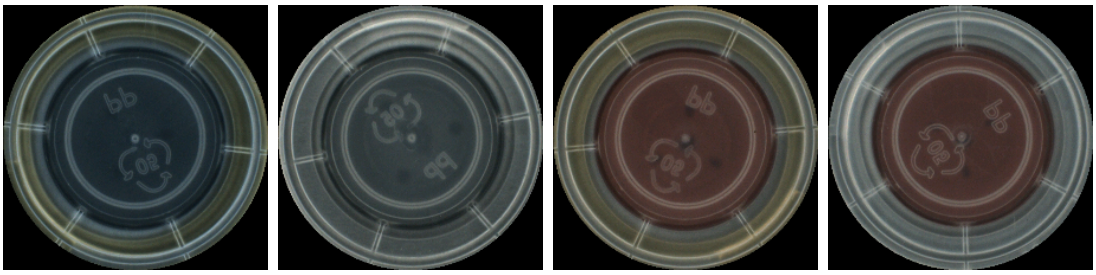
Finally, a Graphical User Interface is created using the "*tkinter*" library. The GUI allows the user to interactively visualize the project results with the addition of graphics and explanations for each algorithm.

5.3 Experiment Results

The task proposed by Antares Vision S.p.A. comprehends images belonging to 9 different categories, so the categories are:

0. Black-Gold Transparent 250 ml, (134 images);
1. Black-Gold Transparent 1000 ml, (6 images);
2. Red-Gold Transparent, (20 images);
3. Red-Silver Transparent, (5 images);
4. Black-Silver Transparent, (20 images);
5. Aluminium, (7 images);
6. Orange, (43 images);
7. Blue, (50 images);
8. Orange with text, (77 images);

for a total of 362 images. The classification is performed using the models: Linear and 3rd degree Support Vector Machine, Decision Tree, Random Forest and K-Nearest Neighbors, but the accuracies result to be not satisfying. From this first phase of analyses, is clear that the classification considering all the classes cannot be handle properly by these algorithms. Taking a closer look at the results is easy to note that all the models fail to recognize the different types of transparent flip-off. These are too similar to be recognized in the correct class, for example the "Black-Gold Transparent" flip-of are slightly different from the "Black-Silver Transparent" or the "Red-Gold Transparent", see figure 5.4.



(a) "Black-Gold" (b) "Black-Silver" (c) "Red-Gold" (d) "Red-Silver"

Figure 5.4: Examples of Transparent Flip-offs.

To face this issue, a different approach is developed: the problem is divided into 2 sub-problems, as explained in chapter 5.1 and shown by the pie plots in figure 5.2. The first sub-problem considers only the transparent flip-offs (as shown in red in the bulleted list above) in order to focus the attention on the details that characterize this type of images. Moreover, in this case the classes "Black-Gold Transparent 250 ml" and "Black-Gold Transparent 1000 ml" are united in an unique category called

"Black-Gold Transparent" because refers to the same type of closure cap. These ampoules can contain different dosages of product, but this is not visible from the images. The second sub-problem groups all the types of transparent flip-offs in an unique class, as shown in blue in the bulleted list above.

- Black-Gold Transparent 250 ml;
- Black-Gold Transparent 1000 ml;
- Red-Gold Transparent;
- Red-Silver Transparent;
- Black-Silver Transparent;
- Aluminium;
- Orange;
- Blue;
- Orange with text.

Black-Gold

Transparent

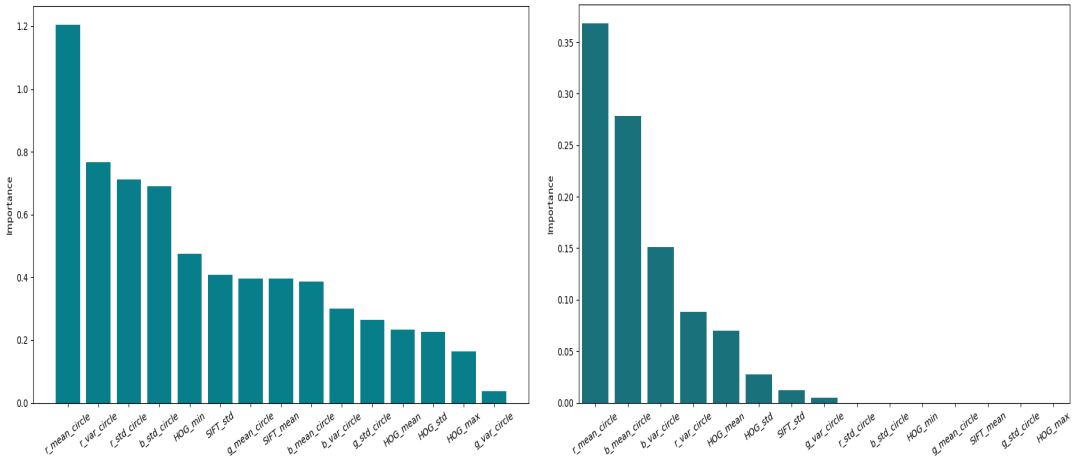
In both cases, embedded feature selection algorithms, in particular a Lasso regression and a tree-based selection, are performed to decide which are the best features to use for classification of the sub-problems. In this way, redundant features are discarded, the training of the algorithms accelerates and improves in term of complexity and computational cost. The first way to access feature importance is by examining directly the coefficients of a regression model in absolute value, after standardizing all the feature on the same scale. The larger the coefficient, more influence that feature will have on the prediction. The second method is a gradient boosting decision tree algorithm, called "XGB tree", that exploits the Information Gain to access the feature importance, as explained in chapter 3.1.3.

Considering only the transparent flip-offs, the features chosen looking on the results of the two embedded feature selection, see figure 5.5, algorithms are:

- Red and Green means;
- Blue standard deviation;
- Red variance;
- HOG minimum;
- SIFT standard deviation.

For the second sub-problem, where the transparent flip-offs are grouped in a unique class, the features chosen looking on the results of the two embedded feature selection, see figure 5.6, algorithms are:

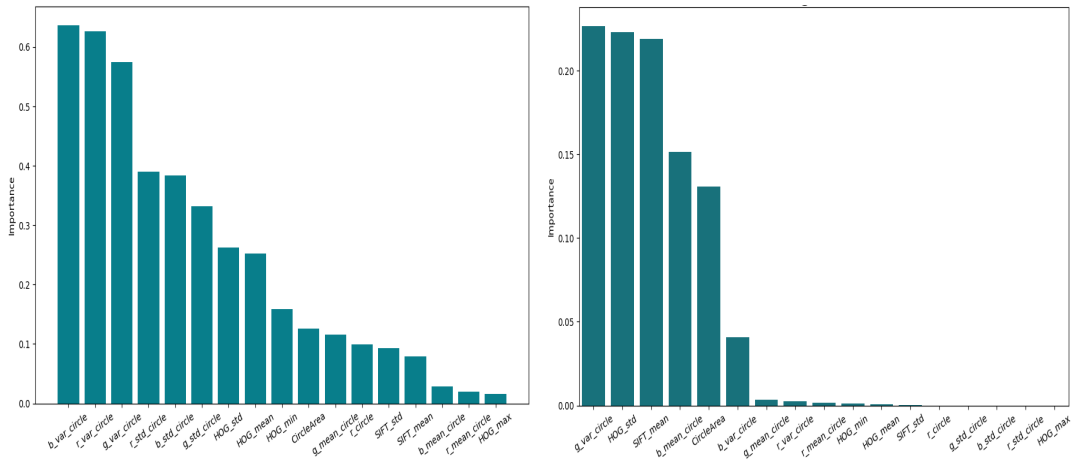
- RGB variances;



(a) Linear regression. (b) XGB tree classifier.

Figure 5.5: Feature ranked by importance for only transparent flip-offs.

- HOG standard deviation;
- Circle Area;
- SIFT mean.



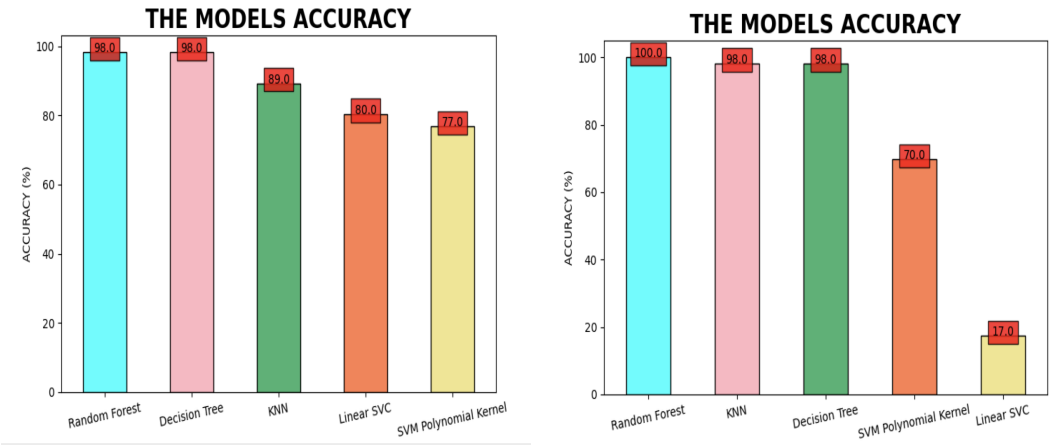
(a) Linear regression. (b) XGB tree classifier.

Figure 5.6: Feature ranked by importance for the case of transparent flip-offs grouped in an unique class.

Note that the ranking created by the linear regression and the XGB tree model are not strictly contemplated, but it is up to the experience of the scientist to use them to choose what he or she believes to be the best features.

Now, for both the sub-problems, the Machine Learning algorithms previously cited are applied using only the features in the lists. The accuracies of the models are shown in figure 5.7. In both cases the Random Forest and the Decision Tree classifiers seem to overperform the Linear and 3rd degree Support Vector Machines.

It should be specified that the results may vary slightly depending on the random creation of the training and validation sets, but the conclusions do not change.

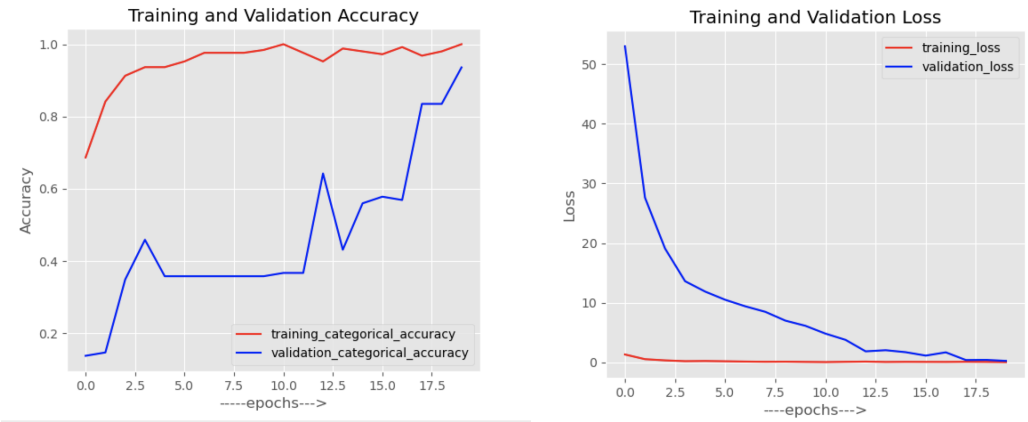


(a) Only transparent flip-off. (b) Transparent flip-off grouped.

Figure 5.7: Models accuracy barplots.

Looking at the results, the approach of dividing the task in two different sub-problems turns out to be performing. However, the initial request was to classify the flip-offs considering all the 9 categories. For that reason, the experiment goes even further and pushes the edges of the scientific research implementing a Convolutional Neural Network, or CNN, and a Vision Transformer for the classification. Both these models do not take as input the features extracted, but directly the labeled images, as explained in chapters 4.6 and 4.7.

The results of the CNN implemented for the resolution of the proposed task are resumed in table 5.1. As shown in the graphs 5.8a, the CNN achieves an accuracy of 90% in 20 epochs, that corresponds to about 3 minutes. Note that the "accuracy"



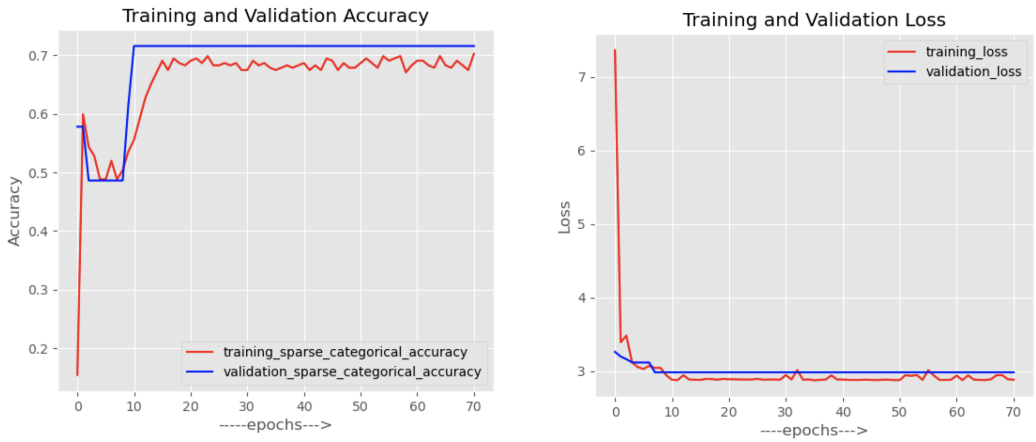
(a) Accuracy plot. (b) Loss plot.

Figure 5.8: CNN plots for achieving an accuracy of 90% in 20 epochs.

is the measure of all the correctly identified cases in percentage. It is calculated

dividing the number of correct prediction by the number of total prediction. The gap between the training and the validation accuracy could indicate an overfitting. This may be caused by the size of the dataset, despite the Data Augmentation performed to compensate this issue. In the loss graphs 5.8b, the learning rate for the validation set seems to be good, so the algorithm has a good convergence speed. Note that the "loss" is a metric that quantifies the difference between the expected outcome and the outcome predicted by the model. Unlike accuracy, loss is not a percentage. It is a summation of the errors made for each sample in training or validation sets.

From the table 5.2 is easy to notice that the Vision Transformer is particularly performant for achieving an accuracy of 60% in just 9 epochs, that corresponds to about 1 minute. As shown in figure 5.9a, the Vision Transformer faces difficulties in achieving the 70% of accuracy and takes about 70 epochs oscillating just above the 70%. This could be an optimization problem of the loss function. Maybe the learning rate is too large and the local minimum of the loss function is missed by the algorithm. Note that the optimization function used for the Vision Transformer is the "Adam" optimizer that has a default learning rate of 0.001. Actually, the Adam optimizer algorithm is an extension of the Stochastic Gradient Descent algorithm that decreases the size of the learning rate getting closer to the minimum. Nevertheless, this algorithm seems to be not very performant for this task, especially figure 5.9b suggests a high learning rate.



(a) Accuracy plot.

(b) Loss plot.

Figure 5.9: Transformers plots for achieving an accuracy of 70% in 71 epochs.

Chapter 6

Conclusions

In this dissertation, a wide selection of algorithms for Computer Vision tasks are reviewed, in particular for the classification of a set of labeled images in several categories. Starting with the image preprocessing, several filters for blurring and smoothing are explored in order to improve the quality of input images, see chapter 1. In addition, the images are described automatically by detecting objects boundaries and searching particular shapes using Edege Detection and Hough Transform algorithms. Data Augmentation is another fundamental step for achieving consistent and reliable analysis. This algorithm consists of increasing the amount of input images in order to face the lack of data problem and avoid a possible overfitting of the training.

The qualities and the characteristics of the images are identified by feature extraction techniques that transform raw data into more manageable information, see chapter 2. In this way, the original dataset is described in an accurate and complete way, but the algorithm reduces the amount of redundant data in order to speed up the analysis. The structure of an object present in an image is described by the Histogram of Oriented Gradients, or HOG, technique with a feature vector of the intensity gradients and the edge directions. Moreover, the local feature in an image that are invariant to transformations are detected with the Scale Invariant Feature Transform, or SIFT. Finally, the surface of the object represented in an image is described as a function of spatial variation of the pixels brightness intensity by the Haralick Texture algorithm.

Another important step to ensure the analysis success is the feature optimization, see chapter 3. These techniques discard unnecessary features, conglomerate and transmute raw features into new ones. The aim is to find the set of features that summarizes the characteristics of the data in the simplest and most accurate way.

The features extracted are exploited for a challenging task: the classification. Different models, called "classifiers", are illustrated in order to categorize the images in the right class with a certain probability, see chapter 4.

The first algorithm used is the Decision Tree classifier. This is a tree structure flowchart build with a sequence of rules that exploits the features extracted to divide the data in classes. Decision Trees are easy to understand and visualize, see image 4.1, but complex trees are prone to overfitting and instability, because small variations in the data can result in completely different results.

In order to improve the Decision Tree accuracy and control the overfitting, a Random Forest classifier is implemented. This model is composed of a number of Decision Trees, five in this case, that are built on different random subsets of the dataset and attributes. On the other hand, this algorithm is more time consuming and complex to interpret and visualize.

Moreover, a K-Nearest Neighbors, or KNN, classifier is computed based on a majority vote scheme of the K nearest neighbors of each data point. Note that $K = 3$ is chosen for this specific task. This is one of the simplest algorithm to implement and understand. It results to be efficient especially for datasets not large in size, where the computational costs and memory storage do not compromise the performance, as in this case.

Additionally, two Support Vector Machine, or SVM, classifiers are implemented: one linear and one polynomial of third degree. The SVMs try to divide the data points belonging to different classes in a N-dimensional space with a hyperplane. When new images are mapped into that space, the prediction to belong to a category is based on which side of the hyperplane they result to be in. SVMs are computational efficient in terms of memory but underperform if the number of features is much greater than the number of samples and is not very robust to noises.

Finally, the experiment pushes the edges of the scientific research implementing a Convolutional Neural Network, or CNN, and a Vision Transformer for the classification. The CNN is a class of Artificial Neural Network and is used to analyze visual imagery, as explained in chapter 4.6. The layers of the CNN are in charge of the feature extraction. The role of the Convolutional NNs is to reduce the images into a form which is easier to process, without losing features which are critical for getting a good prediction. Transformers are efficient Neural Network architectures that exploit the so called "Attention Mechanisms". In recent years only, this technique has been applied to Computer Vision field for image classification.

The challenge proposed in the last chapter consists in the classification of images of closure caps of pharmaceutical ampoules captured by the Inspection Machine department of Antares Vision S.p.A. [65]. The project involves algorithms of image preprocessing, feature extraction and feature engineering in order to end up with the creation of several models for the classification of the images provided by the company. Finally, the models are analyzed and compared to understand which is the best approach to handle this specific problem. A multitude of algorithms is analyzed because is not possible to determine the best model for the given dataset of visual imagery a priori, indeed the results strictly depend on various features that characterize the data. Note that the dataset is labeled, and the classification is based on 8 different categories, see chapter 5.

For this task, several models are implemented, but the unsatisfactory results led to the development of a new approach, that consists of the creation of two subproblems, as explained in chapter 5.3. The classification is performed firstly only on the transparent flip-offs and than on all the other types of flip-offs, grouping all the transparent in a unique class. The classifiers used are: a Linear and a 3rd degree Support Vector Machine, a Decision Tree, a Random Forest and a K-Nearest Neighbor. From the results resumed in table 6.1 is clear that the Random Forest and the Decision Tree classifiers overperform the Linear and 3rd degree Support

Vector Machines.

Models:	Only transparent:	Transparent grouped:
Random Forest	98%	100%
Decision Tree	98%	98%
KNN	89%	98%
Polynomial SVM	77%	70%
Linear SVM	80%	17%

Table 6.1: Accuracies achieved by the models for the two sub-problems.

In order to classify the flip-offs considering all the 9 categories, as required by the initial problem, a Convolutional Neural Network, or CNN, and a Vision Transformer are implemented for the classification. Their results are resumed respectively in tables 5.1 and 5.2. As explained in chapter 5.3, the accuracy and loss graphs show that the CNN performs quite well, despite it faces a bit of overfitting. The Vision Transformers seems to have some issues in exceeding the 70% of accuracy. Some forthcoming enhancements regarding this issue could be:

- the pre-train of the model on several, large datasets, as proposed in the paper [64] written by Bazi *at al.* in 2021. "Pre-train" is a simple technique that consists of first training the Neural Network on a new dataset and then using the parameters founded as a starting point for the actual training of the model on the dataset given for the initial problem. In this way, the Neural Network weights are not randomly initialized and the model optimization is significantly faster. Note that, the dataset used for the pre-training should not be from a completely different domain, if so the results achieved by the pre-trained model could be not so much better.

- From the figure 5.9 is easy to note that the Neural Network faces issues in finding the convergence point of the loss function. This may be caused by a too high learning rate, as suggested by the loss plot in figure 5.9b and the oscillating accuracy shown in figure 5.9a. More in-depth studies about the Vision Transformer algorithm optimization should be performed to face this problem.

Bibliography

[1] Journal of healthcare engineering. https://ietresearch.onlinelibrary.wiley.com/journal/20533713.

[2] Orlando health winnie palmer hospital for women and babies. https://www.orlandohealth.com/strategicinnovations/news-and-stories/gauss-story-with-full-header.

[3] John McCarthy, Marvin L Minsky, Nathaniel Rochester, and Claude E Shannon. A proposal for the dartmouth summer research project on artificial intelligence, august 31, 1955. *AI magazine*, 27(4):12–12, 2006.

[4] Anwaar Ulhaq, Jannis Born, Asim Khan, Douglas Pinto Sampaio Gomes, Subrata Chakraborty, and Manoranjan Paul. Covid-19 control by computer vision approaches: A survey. *Ieee Access*, 8:179437–179456, 2020.

[5] Mohd Javaid, Abid Haleem, Ravi Pratap Singh, Shanay Rab, and Rajiv Suman. Exploring impact and features of machine vision for progressive industry 4.0 culture. *Sensors International*, 3:100132, 2022.

[6] Image pixel representation. https://courses.cs.washington.edu/courses/cse576/book/ch2.pdf.

[7] Luis Perez and Jason Wang. The effectiveness of data augmentation in image classification using deep learning. *arXiv preprint arXiv:1712.04621*, 2017.

[8] Cambridge dictionary. https://dictionary.cambridge.org/it/dizionario/inglese/augmentation.

[9] Connor Shorten and Taghi M Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of big data*, 6(1):1–48, 2019.

[10] Image rotation. https://www.sciencedirect.com/topics/computer-science/image-rotation.

[11] Richard Szeliski. *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

[12] O Rebecca Vincent, Olusegun Folorunso, et al. A descriptive algorithm for sobel image edge detection. In *Proceedings of informing science & IT education conference (InSITE)*, volume 40, pages 97–107, 2009.

[13] John Francis Canny. Finding edges and lines in images. Technical report, MASSACHUSETTS INST OF TECH CAMBRIDGE ARTIFICIAL INTELLIGENCE LAB, 1983.

[14] Richard O Duda and Peter E Hart. Use of the hough transformation to detect lines and curves in pictures. *Communications of the ACM*, 15(1):11–15, 1972.

[15] Dana H Ballard. Generalizing the hough transform to detect arbitrary shapes. *Pattern recognition*, 13(2):111–122, 1981.

[16] Saurabh Agrawal, Nishchal K Verma, Prateek Tamrakar, and Pradip Sircar. Content based color image classification using svm. In *2011 Eighth International Conference on Information Technology: New Generations*, pages 1090–1094. IEEE, 2011.

[17] Robert K McConnell. Method of and apparatus for pattern recognition, January 28 1986. US Patent 4,567,610.

[18] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, volume 1, pages 886–893. Ieee, 2005.

[19] David G Lowe. Object recognition from local scale-invariant features. In *Proceedings of the seventh IEEE international conference on computer vision*, volume 2, pages 1150–1157. Ieee, 1999.

[20] Hui Kong, Hatice Cinar Akakin, and Sanjay E Sarma. A generalized laplacian of gaussian filter for blob detection and its applications. *IEEE transactions on cybernetics*, 43(6):1719–1733, 2013.

[21] Cambridge dictionary. `https://www.oxfordlearnersdictionaries.com/definition/american_english/texture`.

[22] Robert M. Haralick, K. Shanmugam, and Its'Hak Dinstein. Textural features for image classification. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-3(6):610–621, 1973.

[23] Dong-Chen He and Li Wang. Texture unit, texture spectrum, and texture analysis. *IEEE transactions on Geoscience and Remote Sensing*, 28(4):509–512, 1990.

[24] Timo Ojala, Matti Pietikäinen, and David Harwood. A comparative study of texture measures with classification based on featured distributions. *Pattern recognition*, 29(1):51–59, 1996.

[25] Artur J Ferreira and Mário AT Figueiredo. Efficient feature selection filters for high-dimensional data. *Pattern recognition letters*, 33(13):1794–1804, 2012.

[26] John T Kent. Information gain and a general measure of correlation. *Biometrika*, 70(1):163–173, 1983.

[27] Huan Liu and Rudy Setiono. Chi2: Feature selection and discretization of numeric attributes. In *Proceedings of 7th IEEE International Conference on Tools with Artificial Intelligence*, pages 388–391. IEEE, 1995.

[28] Zhixiang Xu, Gao Huang, Kilian Q Weinberger, and Alice X Zheng. Gradient boosted feature selection. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 522–531, 2014.

[29] Lilli Haar, Katharina Anding, Konstantin Trambitckii, and Gunther Notni. Comparison between supervised and unsupervised feature selection methods. In *ICPRAM*, pages 582–589, 2019.

[30] Max Kuhn, Kjell Johnson, et al. *Applied predictive modeling*, volume 26. Springer, 2013.

[31] Max Kuhn and Kjell Johnson. *Feature engineering and selection: A practical approach for predictive models*. CRC Press, 2019.

[32] Girish Chandrashekar and Ferat Sahin. A survey on feature selection methods. *Computers & Electrical Engineering*, 40(1):16–28, 2014.

[33] Fengxi Song, Zhongwei Guo, and Dayong Mei. Feature selection using principal component analysis. In *2010 international conference on system science, engineering design and manufacturing informatization*, volume 1, pages 27–30. IEEE, 2010.

[34] Alice Zheng and Amanda Casari. *Feature engineering for machine learning: principles and techniques for data scientists*. " O'Reilly Media, Inc.", 2018.

[35] Franziska Horn, Robert Pack, and Michael Rieger. The autofeat python library for automated feature engineering and selection. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 111–120. Springer, 2019.

[36] Maximilian Christ, Nils Braun, Julius Neuffer, and Andreas W Kempa-Liehr. Time series feature extraction on basis of scalable hypothesis tests (tsfresh–a python package). *Neurocomputing*, 307:72–77, 2018.

[37] Hoang Thanh Lam, Johann-Michael Thiebaut, Mathieu Sinn, Bei Chen, Tiep Mai, and Oznur Alkan. One button machine for automating feature engineering in relational databases. *arXiv preprint arXiv:1706.00327*, 2017.

[38] Gilad Katz, Eui Chul Richard Shin, and Dawn Song. Explorekit: Automatic feature generation and selection. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pages 979–984. IEEE, 2016.

[39] Oxford dictionary. `https://www.oxfordlearnersdictionaries.com/definition/american_english/classification`.

[40] Aurélien Géron. Hands-on machine learning with scikit-learn and tensorflow: Concepts. *Tools, and Techniques to build intelligent systems*, 2017.

[41] Tin Kam Ho. Random decision forests. In *Proceedings of 3rd international conference on document analysis and recognition*, volume 1, pages 278–282. IEEE, 1995.

[42] Evelyn Fix and Joseph L Hodges. Discriminatory analysis, nonparametric discrimination: consistency properties. us air force school of aviation medicine. *Technical Report 4*, 1951.

[43] Naomi S Altman. An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, 46(3):175–185, 1992.

[44] Jiaxin Zhuang, Jiabin Cai, Ruixuan Wang, Jianguo Zhang, and Wei-Shi Zheng. Deep knn for medical image classification. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 127–136. Springer, 2020.

[45] MA Mukid, T Widiharih, A Rusgiyono, and A Prahutama. Credit scoring analysis using weighted k nearest neighbor. In *Journal of Physics: Conference Series*, volume 1025, page 012114. IOP Publishing, 2018.

[46] Sadegh Bafandeh Imandoust, Mohammad Bolandraftar, et al. Application of k-nearest neighbor (knn) approach for predicting economic events: Theoretical background. *International journal of engineering research and applications*, 3(5):605–610, 2013.

[47] Sara Haddou Bouazza, Nezha Hamdi, Abdelouhab Zeroual, and Khalid Auhmani. Gene-expression-based cancer classification through feature selection with knn and svm classifiers. In *2015 Intelligent Systems and Computer Vision (ISCV)*, pages 1–6. IEEE, 2015.

[48] V Vapnik and A Ya Lerner. Recognition of patterns with help of generalized portraits. *Avtomat. i Telemekh*, 24(6):774–780, 1963.

[49] Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152, 1992.

[50] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[51] Peijun Du, Junshi Xia, Wei Zhang, Kun Tan, Yi Liu, and Sicong Liu. Multiple classifier system for remote sensing image classification: A review. *Sensors*, 12(4):4764–4792, 2012.

[52] Luís Gonçalves. Multiple classifier systems — a brief introduction. https://medium.com/luisfredgs/multiple-classifier-systems-a-brief-introduction-71238d9c794f, 2021.

[53] Tin Kam Ho, Jonathan J. Hull, and Sargur N. Srihari. Decision combination in multiple classifier systems. *IEEE transactions on pattern analysis and machine intelligence*, 16(1):66–75, 1994.

[54] Julian Fierrez, Aythami Morales, Ruben Vera-Rodriguez, and David Camacho. Multiple classifiers in biometrics. part 1: Fundamentals and review. *Information Fusion*, 44:57–64, 2018.

[55] Grace W Lindsay. Convolutional neural networks as a model of the visual system: Past, present, and future. *Journal of cognitive neuroscience*, 33(10):2017–2031, 2021.

[56] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[57] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[58] Aleksei Grigorevich Ivakhnenko, A G Ivakhnenko, Valentin Grigorevich Lapa, and Valentin Grigorevich Lapa. *Cybernetics and forecasting techniques*, volume 8. American Elsevier Publishing Company, 1967.

[59] Christoph Molnar. A guide for making black box models explainable. *URL: https://christophm. github. io/interpretable-ml-book*, 2018.

[60] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[61] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[62] Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. Image transformer. In *International conference on machine learning*, pages 4055–4064. PMLR, 2018.

[63] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[64] Yakoub Bazi, Laila Bashmal, Mohamad M Al Rahhal, Reham Al Dayil, and Naif Al Ajlan. Vision transformers for remote sensing image classification. *Remote Sensing*, 13(3):516, 2021.

[65] Antares Vision S.p.A. `https://it.antaresvision.com`.

[66] Elisa Salvi. Internship project Antares Vision S.p.A. `https://github.com/SalviElisa98/stage_project_py`, 2022.

[67] Documentation OpenCV. `https://docs.opencv.org/4.x/dd/d1a/group__imgproc__feature.html#ga47849c3be0d0406ad3ca45db65a25d2d`.

Acknowledgements

First and foremost, I would like to thank my thesis advisor, Dr. Enrico Barbierato of the Mathematics department at Sacred Heart Catholic University of Brescia, for his invaluable patience and feedback. I must express my very profound gratitude for his passionate participation and enthusiasm for the topic. Without his constant assistance and dedicated involvement in every step throughout the process, this work would have never been successfully accomplished. Additionally, this dissertation would not have been possible without the valuable support of my co-advisor, Professor Alfredo Marzocchi, who kindly assisted me and steered me in the right direction.

In March 2022, I had the opportunity to join the Inspection Machine department of Antares Vision S.p.A. for several weeks to develop the project presented in chapter 5. My time there has been highly productive and working with Antares Vision software developers was an extraordinary experience. I would like to show gratitude to the project manager Alberto Reghenzi and his committee, who generously provided me knowledge and expertise, especially Emanuela, Simone, Davide, Paolo, Nicola, Flavio and all the others.

Most important, words cannot express my gratitude to my family, especially my parents. We have experienced difficult moments in the past two years, but their unconditional love and encouragement stand as a testament for the success of this work. Thank you.

⟨*Elisa Salvi*⟩