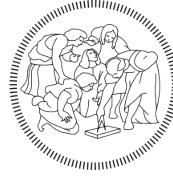


POLITECNICO
MILANO 1863

Politecnico di Milano

School of Industrial and Information Engineering

Master of Science in Mathematical Engineering - Statistical Learning

Lung Cancer Cell Images Classification by Fisher Vector and Convolutional Neural Network Features

Supervisor: Prof. Giacomo Boracchi
Co-supervisor: Prof. João Batista Florindo

Candidate:
Giacomo Ruffoni Matr. 899361

Academic Year 2019-2020

Abstract

In the world of public health, one of the most critical problems about lung cancer is making fast and good diagnosis. This is a task that can be efficiently solved by *machine learning* image classifications techniques. In this work we use a private dataset consisting of lung cancer cells images divided in four classes: adenocarcinoma, epidermoid carcinoma, OAT-cells and negative cells. To solve the classification task, we first extract powerful images features truncating a *Convolutional Neural Network* still at level of convolutional layers, training it from scratch or using transfer learning with fine-tuned parameters, both with and without applying *data augmentation*. Images features are then pooled by applying different types of pooling encoders, like *Fisher Vector*, *Bag of Visual Words* and *Vector of Locally Aggregated Descriptors*. This is done in order to make them more suitable to be classified by a linear *Support Vector Machine*, which is trained with the pooled features. The experiments show that, using data augmentation and fine-tuning the parameters of a pre-trained CNN, the results obtained with FV encoding quite overtake all the other tested techniques (best mAP: 89.8% train, 70.4% test). This work demonstrates that the above mentioned structure reaches competitive performance in solving medical image classifications tasks.

Sommario

Per la sanità pubblica, uno dei più grandi problemi riguardanti il cancro ai polmoni è riuscire ad effettuare rapidamente diagnosi corrette. Questo compito può essere risolto in modo efficiente mediante tecniche di classificazione di immagini con *machine learning*. In questo lavoro è stato utilizzato un dataset privato, costituito da immagini di cellule di cancro ai polmoni suddivise in quattro classi: adenocarcinoma, carcinoma epidermoide, cellule OAT e cellule negative. Per classificarle, vengono innanzitutto estratte le caratteristiche delle immagini troncando una *Rete Neurale Convoluzionale* a livello degli strati convoluzionali, sia allenandola da zero e sia utilizzando *transfer learning* ottimizzando i parametri, in entrambi i casi con e senza *data augmentation*. Le caratteristiche estratte vengono quindi raggruppate applicando diversi tipi di *pooling encoders*, come *Fisher Vector*, *Bag of Visual Words* e *Vector of Locally Aggregated Descriptors*. Questi hanno la funzione di renderle più adatte ad essere classificate da una *Macchina a Vettori di Supporto* lineare, la quale viene poi allenata proprio con le caratteristiche raggruppate. Gli esperimenti mostrano che, utilizzando *data augmentation* e ottimizzando i parametri di una RNC già allenata, i risultati ottenuti usando FV come encoder superano ampiamente tutte le altre tecniche testate (mAP migliore: 89.8% train, 70.4% test). Questo lavoro dimostra che la struttura sopra menzionata raggiunge prestazioni competitive nella risoluzione dei compiti di classificazione di immagini mediche.

Contents

Abstract 3

Sommario 5

List of Figures 11

List of Tables 13

Introduction 15

1 Machine Learning Background 19
- 1.1 Unsupervised Learning . . . 19
  - 1.1.1 Clustering Algorithms . . . 20
- 1.2 Supervised Learning . . . 23
  - 1.2.1 Artificial Neural Networks . . . 24
  - 1.2.2 Support Vector Machine . . . 30
  - 1.2.3 Problem of Imbalanced Classes . . . 36

2 Image Classification Problem 39
- 2.1 Medical Image Classification . . . 40
  - 2.1.1 Lung Cancer Detection . . . 41
- 2.2 Problem Formulation . . . 42
  - 2.2.1 Related Works . . . 44

3 Image Features: Extraction and Pooling 47
- 3.1 Convolutional Neural Networks . . . 47
  - 3.1.1 The Convolution Operation . . . 48
  - 3.1.2 General Architecture . . . 48
  - 3.1.3 Learning Techniques . . . 55
- 3.2 Pooling Encoders . . . 59
  - 3.2.1 Fisher Vector . . . 59
  - 3.2.2 Bag of Visual Words . . . 65
  - 3.2.3 Vector of Locally-Aggregated Descriptors . . . 69

4 Proposed Solution 71

5 Experiments 79

5.1 Dataset . . . 79

5.2 Experiment 01 . . . 81

5.3 Experiment 02 . . . 87

Conclusions 89

Bibliography 90

Ringraziamenti 97

List of Figures

1 Percentage of lung cancer cases with respect to all the lethal cancers in the EU states in 2015. Image taken from [4]. 15

2 Examples of lung cells for each class in the dataset. 16

3 Basic structure of the classification learning process implemented in this work. The features are extracted still at the level of convolutional layers, instead of right before the last fully-connected one. In particular, this is what gives the big amount of local descriptors for each image. 17

1.1 Unsupervised Learning simple block scheme. 19

1.2 DBSCAN example with 2 clusters (taken from [2]). 20

1.3 Dendrogram example of a 6 elements hierarchical agglomerative clustering. It joins at each step the two nearest clusters according to the chosen linkage criteria. 22

1.4 Supervised Learning simple block scheme. 23

1.5 Generic structure of a 1-hidden layer ANN with 9 total neurons. 25

1.6 Structure of a perceptron with 3 inputs and a bias. x_i are the inputs, w_i their weights, b is the bias and y is its output. 25

1.7 $L : \mathbb{R} \rightarrow \mathbb{R}$ is a simple convex function. Here each red arrow represents the direction of the negative gradient that determines the parameters updating. Note how it gets shorter and shorter approaching the minimum. 29

1.8 2-dimenensional example of a SVM classifier for *linearly separable data*. The red line represents the margin, the black line is the classification hyperplane, i.e. $f(\mathbf{x}) = 0$, while the dashed ones are the support vectors, i.e. $f(\mathbf{x}) = \pm 1$. 31

1.9 Example of observations mapped from a 2D space to a 3D one, where non-linear separation surfaces can become well approximated by linear ones. In real world applications, very high dimensional embedding spaces are used. 34

1.10 Confusion matrix of a generic binary classification problem. "+" and "-" represent the positive and negative class. *TP* and *TN* are respectively the number of positive and negative samples correctly classified, while *FP* and *FN* represent instead the negative and positive misclassified observations. 37
1.11 Simple example of a Precision-Recall curve with 10 positive samples (i.e. in C) and 20 negative ones. Taken from [15]. 38

2.1 Possible generic images inside a dataset. 2.1a, 2.1b and 2.1c represent examples of *Scene* images, while 2.1d, 2.1e and 2.1f are instead examples of *Object* images. 39
2.2 Brain Tumor Images. Taken from [42]. 40
2.3 Hands and Wrist Bone Fractures Images. Taken from [66]. 40
2.4 Intracranial Hemorrhages Images. Taken from [26]. 41
2.5 2.5a: imaging test via CT scan. 2.5b cytologic test via biopsy. 42

3.1 General structure of a convolutonal layer (taken from [6] and slightly modified). The depth of the output depend on the number of channels we set for the layer (hyperparameter). 48
3.2 General structure of a pooling layer (taken from [6] and slightly modified). 50
3.3 2×2 max pooling window with stride 2, on a 4×4 input image. It partitions the input image into a set of non-overlapping squares[1]and, for each sub-region, outputs the maximum. 51
3.4 Heaviside 52
3.5 Sigmoid 52
3.6 ReLU 52
3.7 Here are shown the connections in a simple neural network structure. (a) Without using the dropout layer. (b) Using the dropout layer. . . . 53
3.8 Flattening and fully connected layer with two final classes. 54
3.9 Example (taken from [5]) in 2 dimensions of K-means algorithm, with $K = 3$ and random initialization of the centroids. Note how the three centroids change position during the process. 66
3.10 In this image (taken from [1]) it is shown how the BoVW works: first the features are extracted and then each image is represented as a frequency histogram of its features. Here the constructed vocabulary has $K = 4$. . 67

4.1 Detailed scheme of the proposed classification process. After cropping the images, the features are extracted. An encoder is then applied in order to obtain the vectors of pooled features (we write "FV Pooling" since it is the pooling encoder we propose to use). Finally, the vectors are passed to a linear SVM to be classified. 71

4.2 Images pre-processing when training *MyNet* from scratch and when fine-tuning *VGG-m* parameters. The first consists in augmenting the input image, the second one is the subtraction of the average training image. . 72
4.3 *MyNet* architecture used during training. 72
4.4 *VGG-m* architecture used during training. Note that the las fully connected layer outputs only 4 classes and not 1000, as it is in its original framework. 73

5.1 Examples of lung cells for each class in the dataset. 80
5.2 *Loss Function*: log-loss, the value continues to decrease but without any evident improvement. 81
5.3 Feature map of a random chosen cell extracted after 9th layer of *MyNet*. 81
5.4 *MyNet* trained from scratch. APs of the four classes using FV encoder. 82
5.5 5.5a: total acc = 81.1%. 5.5b: total acc = 66.0%. 83
5.6 Feature maps of a random chosen cell extracted after 13th layer of *AlexNet* (5.6a) and *VGG-m* (5.6b). 83
5.7 APs of the four classes using FV. 84
5.8 *Loss Function*: log-loss, the value continues to decrease for training, but stops and starts oscillating for test. 85
5.9 *VGG-m* fine-tuned. APs of the four classes using FV encoder. 86
5.10 5.10a: total acc = 89.5%. 5.10b: total acc = 69.5%. 86
5.11 APs for the two augmented dataset using FV. 88

List of Tables

4.1 *MyNet* architecture. The separation line between the 9th and 10th layers indicates where we extract the image features before applying the encoders. 75
4.2 *AlexNet* architecture. The separation line between the 13th and 14th layers indicates where we extract the image features before applying the encoders. 76
4.3 *VGG-m* architecture. The separation line between the 13th and 14th layers indicates where we extract the image features before applying the encoders. 77

5.1 Train and test mAP values (in %) when extracting features with *MyNet*. 82
5.2 Train and test mAP values (in %) when extracting features with *AlexNet*. 84
5.3 Train and test mAP values (in %) when extracting features with *VGG-m*. 84
5.4 Train and test mAP values (in %) when extracting features with *VGG-m* fine-tuned from the 9th layer to the last one. 85
5.5 Train and test mAP values (in %) when extracting features with *VGG-m*. 87

Introduction

Cancer globally represents one of the major problems in the world of public health. In 2013 ([9]) it was estimated that worldwide there were 14.9 million incident cancer cases and 8.2 million deaths. Of all the types of cancer, the one with the highest number of deaths is the lung cancer, with 1.6 million deaths (20% of the total deaths) and an estimated annually growth of 1.8 million cases. Just in the European Union, around 312,000 people (213,000 men and 99,000 women) are diagnosed with lung cancer every year. This makes lung cancer the second and third most commonly diagnosed cancer in men and women, respectively ([38]). In terms of economic costs, lung cancer is the first and represents the 15% of the total cancer cost in EU, with estimated 18.8 billions in 2009 ([37]). Nowadays the needs and the challenges in this area are still multiple, touching fields as patients rights, treatment cost, early diagnosis and tumor detection techniques. In the last decade, many improvements have been made and patients life quality has drastically increased, thanks also to more safe and effective therapies and diagnostic tools.

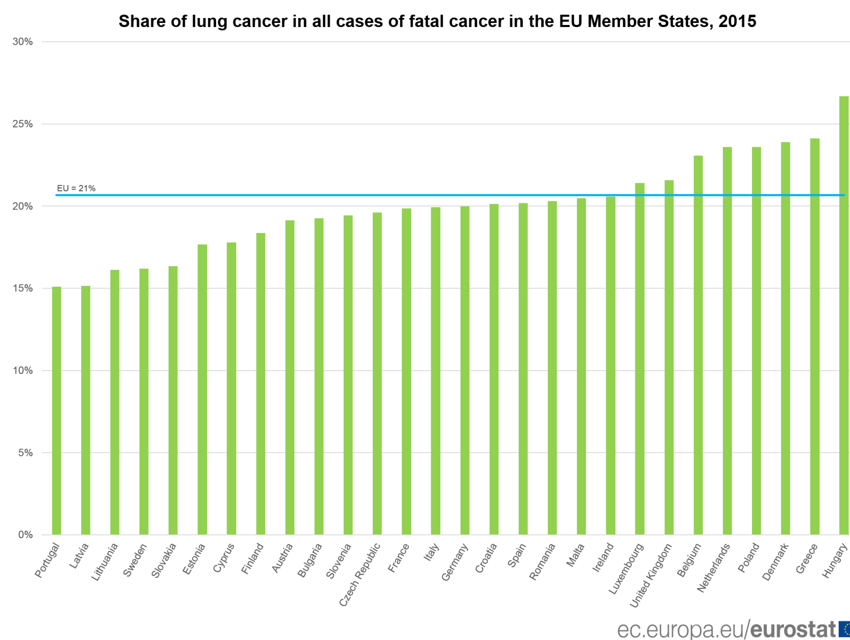
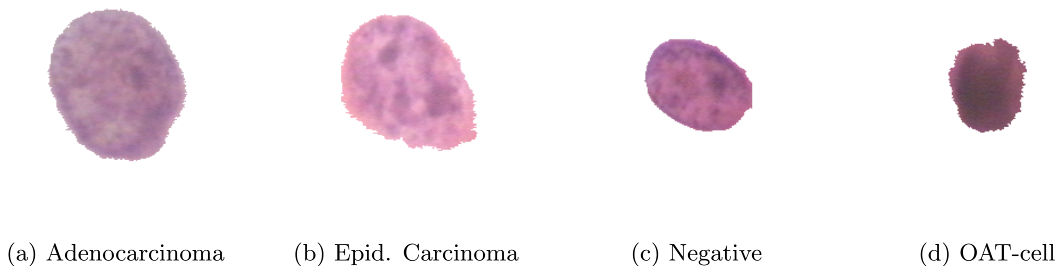


Figure 1: Percentage of lung cancer cases with respect to all the lethal cancers in the EU states in 2015. Image taken from [4].

There exist different types of lung cancers and a first categorization, both in terms of danger and localization in the lung, is between *Non-Small Cell* and *Small Cell* ones. To accurately give a diagnosis on the type of cancer, a pathologist has to take into account many useful features that the cells may or may not present. Making fast and good diagnosis represents often a problem and reducing the public health total cost of lung cancer results difficult with an increasing number of cases per year arounf the world. In particular, recognizing the lung cancer "patterns" in a given cell image requires a "learning from experience" approach. The structure of this task is well suited to be tackled by *machine learning* techniques, which could easily reduce both time and cost of diagnoses. As a consequence, in order to do that efficiently, a world of ideas and approaches opens up in front of the scientific community and several researchers have studied and developed many different learning techniques to face and solve this problem.

The dataset used for this analysis is not public and was kindly offered by a collaboration with the *Department of Pathological Anatomy, Faculty of Medical Sciences* and the *Faculty of Mathematics, Statistics and Scientific Computation* of University of Campinas (UNICAMP), Brazil. As typically happens when collecting medical data, it does not contain many samples. In particular, it is composed of 5277 lung cell images divided in four imbalanced classes that indicate the presence of the tumor and, in the affirmative case, also its type. The four classes are *Adenocarcinoma, Epidermoid Carcinoma, Negative* and *OAT-cell*. Examples of images are shown in Figure 2.



(a) Adenocarcinoma (b) Epid. Carcinoma (c) Negative (d) OAT-cell

Figure 2: Examples of lung cells for each class in the dataset.

The approaches that first tried to efficiently classify medical images were based on shallow methods, which take into account low-level features of the images, relying on things like colour, shape or texture. These traditional methods were performed directly classifying the extracted features making use of techniques like random forests ([48]) or support vector machines (SVMs) ([67]). The weaknesses of low-level features are their poor ability to generalize to interpret high-level domain problems. New well-performing techniques exploit instead the concept of deep-learning architectures, which has the capability of detecting high-level features and can compute final classification labels of the input images. The deep architectures that have been proven to be the best

in solving image classification tasks are the *Convolutional Neural Networks* (CNNs), a specific type of neural networks that is able to take into account the position of features inside the images. Their use is not just limited to classification, but also to other related problems like, for example, segmentation ([49]). The CNN structure is able to perform classification by itself, but many researchers used it, as with SIFT, to extract the important features of the images and then pass them to a SVM classifier ([8]). The CNN's built in classifier allows to perform training end-to-end and, in general, is preferred to the SVM, since the latter often does not improve significantly the performance of the classification. This is why experts usually think that it is not worth to use it. The cause of this lack in performance improvement must be sought in the fact that high-level features extracted with the CNN are not suitable to be well-interpreted by a SVM, which, as a consequence, is not able to learn how to classify images.

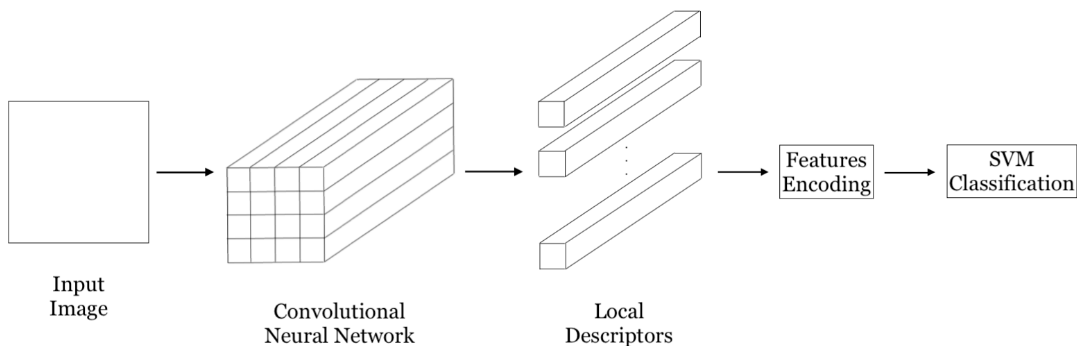


Figure 3: Basic structure of the classification learning process implemented in this work. The features are extracted still at the level of convolutional layers, instead of right before the last fully-connected one. In particular, this is what gives the big amount of local descriptors for each image.

To overcome this problem, we introduce, following the work of Cimpoi et al. ([19]), a method called *Fisher Vector* (FV) that could be defined as a specific *Pooling Features Encoder*. Here the features extracted from the CNN are called local descriptors and can be seen as a sort of *banks* of non-linear filters. The FV allows the SVM classifier to better interpret the information contained in these local descriptors, feeding it with a more precise images representation than the one given directly by the CNN. In Figure 3 is shown how the pooling encoders, specifically the FV, intervene in the classification process. We will also show that FV performance improvement quite overtakes also two other types of pooling encoders: *Bag of Visual Words* (BoVW) and *Vector of Locally Aggregated Descriptors* (VLAD). Moreover, two techniques to obtain local descriptors from the CNN are tested. The first one consists in the *Training from Scratch* of the network, in which the CNN parameters are randomly initialized and then trained according to the target dataset. The second approach is the so-called *Transfer Learning*,

which exploits the fact that in literature there exist networks that have already been trained on huge amount of data and could be reused to efficiently extract features from completely new images. Since in our specific case the available training samples are not so many, the second method is expected to quite overtake the first one. We also exploit two more techniques: *Data Augmentation* and *Parameters fine-tuning*, which, in particular, should be able to reduce overfitting and slightly improve the performance of the experiments.

The main contribution of this thesis concerns the usage of the *Fisher Vector* as a very efficient pooling encoder in solving lung cancer cell images classification problems, where *Convolutional Neural Networks* are used as features extractors and a linear *Support Vector Machine* as classifier. Moreover, we also show that when a small amount of training samples is available, as in the specific analysis on our dataset, the proposed solution can be improved using *Data Augmentation* and *Transfer Learning* techniques, in particular fine-tuning networks parameters. Specifically, we obtained the following improvements: 56.5% test mAP when training CNN from scratch, 70.4% test mAP when using data augmentation and transfer learning with parameters fine-tuning (in both cases using FV pooling and a linear SVM). This classification framework is promising since, when given a small amount of data, the linear SVM is computationally easy to train and it does not tend to overfit (unlike CNN's built in classifier). Moreover, it manages to exploit the high-level features extracted with the CNN and then pooled using FV to perform multi-label classification.

Chapter 1

Machine Learning Background

Machine Learning (ML) is a subfield of *Artificial Intelligence* (AI) that aim to make predictions or decisions. They are able to solve these tasks without being explicitly programmed to do so, but building a mathematical model based on a set of input samples. Inside the ML world there exist many different approaches to do that, depending on the type of data available to the learning system, its structure and also according to the task the algorithm is asked to perform. The most part of these methods can be divided into the following three groups: *Unsupervised Learning*, *Supervised Learning* and *Reinforcement Learning*. We try to give a detailed explication of the first two categories since the latter one is not exploited in this thesis.

1.1 Unsupervised Learning

The *unsupervised* learning is a machine learning technique that tries to divide in classes a set of data that it's not originally divided into labels, i.e. it's not known a priori the group each sample in the dataset belong to. The logical process of this approach is shown in Figure 1.1.

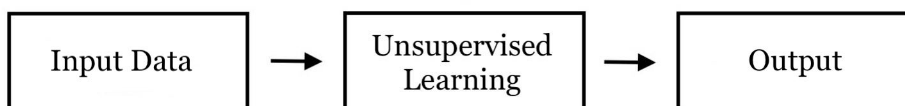


Figure 1.1: Unsupervised Learning simple block scheme.

Generally speaking, the unsupervised learning does not need an enormous quantity of data to give interesting results ([10]), but its performance is lower with respect to the one that could be obtained with labeled data in a supervised learning framework (1.2). To divide the inputs in groups, unsupervised learning techniques try to detect, during processing phase, some common pattern between samples. Depending on the chosen learning algorithm, this is done in different ways.

The two main families of algorithms used to classify data without labels are *Clustering* and *Association Rules.* Here in this thesis, explaining the latter category would be redundant and because of that we are presenting just the first one.

1.1.1 Clustering Algorithms

Clustering is an unsupervised learning technique that aims to group a set of data observations in such a way that the samples in different groups are very different between each others, while those in the same group are very similar. The groups are also called *clusters.* The mathematical evaluation of the notion of "different/similar" is usually related to the distance between the data points, how they are distributed or the dense areas they form in the samples space.
In the world of clustering, there exist many different ways to approach the problem of how to assign samples to clusters, and so how to create them. Let's so present some of the widely used types of cluster models and algorithms.

1.1.1.1 Density-based Clustering

Density-based clustering is a technique based on the idea that the samples space could be divided in clusters regions, in particular *high point density* and *sparse* ones. The two most popular algorithms that exploit this concept are the *Density-Based Spatial Clustering of Applications with Noise* (DBSCAN) and the *Mean-Shift.*

- *DBSCAN:* this algorithm, originally proposed by [20], gathers observations according to some density criterion that has to be satisfied. In the original variant, this criterion is defined as the minimum number of objects within a radius.

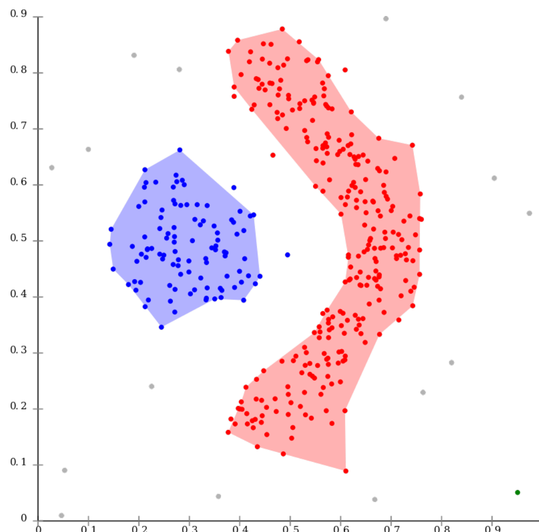


Figure 1.2: DBSCAN example with 2 clusters (taken from [2]).

The main advantages of this method are that it is computationally light, it gives almost always the same result for every run (the only points that are not deter-

ministically assigned to clusters are the *border points*) and it is able to identify clusters of approximately any shape (see Figure 1.2). Its biggest disadvantage is that it expects a density drop to identify the clusters borders.

- *Mean-Shift:* this algorithm is based on *Kernel Density Estimation* (KDE) and follows the procedure of locating the *modes* of a density function (i.e. the chosen kernel). It is an iterative process that starts with an initial point that is updated at each step, moving it in the direction of a specific dense point area. This dense area is evaluated just on the points inside a circle with center in the already mentioned point and a priori fixed radius (also referred to as *bandwidth*). The data observations are assigned to a cluster if they lie inside this circle at least once during the updating process.
  As DBSCAN, mean-shift can detect arbitrary-shaped clusters, but due to an iterative procedure that is computationally very expansive, its convergence is usually slower. Another weakness of this method is that it doesn't perform well in high dimensional samples space. Beside this, mean-shift presents many advantages: the only hyperparameter to set is the bandwidth of the circle, which in practice also indirectly determines the number of clusters; it requires no model assumptions - other than the choice of the kernel - and, thanks to KDE, it is not so affected by outliers, which at most could lead to the creation of singleton clusters. For further details and to see how this algorithm could be improved see [16].

1.1.1.2 Connectivity-based Clustering

Also called *Hierarchical Clustering*, this class of models is based on the idea that the closer two objects are in the samples space, the more probable is that they belong to the same cluster. The two general approaches to perform hierarchical clustering are: *agglomerative*, where at starting point each observation coincides with a cluster and moving up into the hierarchy the clusters are joined, and *divisive*, in which all the objects start in one cluster which is then split during the process. To measure the distance between two observations x and y in the dataset, a *metric* $d(x, y)$ has to be chosen. The most common metrics are the *Euclidian* and the *Manhattan* distances (see (3.36) for their mathematical expressions), but many others could be efficiently used. To decide how to join/split clusters, it is necessary to define how to evaluate "proximity" between them. To do that, the user has to choose one of the so-called *Linkage Criteria*, which involve the use of the already chosen distance. Some of the commonly well-performing criteria are:

- *Single-Linkage Clustering.* To evaluate the proximity between two clusters X, Y, it considers the shortest distance between objects of the clusters:

$$SL(X,Y) = \min\{d(x,y) : x \in X, y \in Y\} \tag{1.1}$$

- *Complete-Linkage Clustering.* Ideally very similar to the first type, this criteria considers the maximum distance between objects:

$$CL(X,Y) = \max\{d(x,y) : x \in X, y \in Y\} \tag{1.2}$$

- *Average-Linkage Clustering.* In this criteria the average of all distances between the elements of the two clusters is performed:

$$AL(X,Y) = \frac{1}{|X| \cdot |Y|} \sum_{x \in X} \sum_{y \in Y} d(x,y) \tag{1.3}$$

To show how the clusters are joined/splitted during the process, it is introduced a diagram representing tree called *dendrogram* (see Figure 1.3).

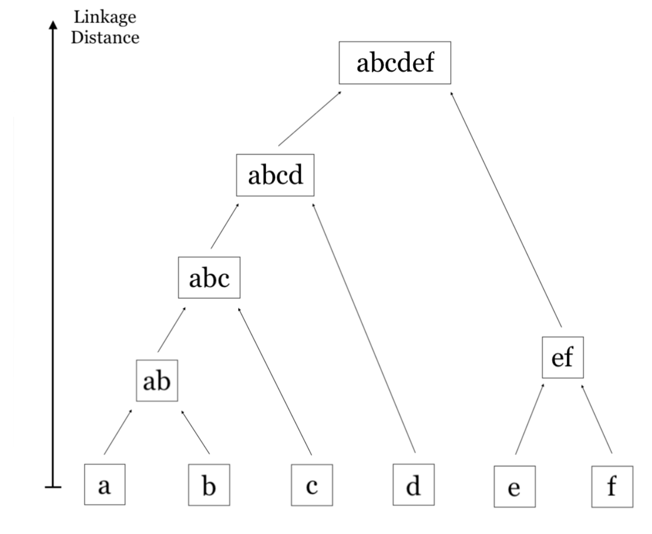


Figure 1.3: Dendrogram example of a 6 elements hierarchical agglomerative clustering. It joins at each step the two nearest clusters according to the chosen linkage criteria.

Since the hierarchical clustering does not give a unique clusters partition, there exist many different evaluation methods that the user should take into account to decide the final number of clusters (e.g. the *Elbow Method*). In *data mining* this clustering model performs very well and is considered the theoretical basis in cluster analysis, but, because of its high computational complexity, it is not so applied in real problems.

1.1.1.3 Centroid-based Clustering

In centroid-based clustering the number of clusters is a priori fixed. The clusters are represented by a vector of *centroids*, one for each cluster. They may or may not be members of the dataset and their initialization is usually done at random. The algorithm that exploits this concept with good approximation and is one of the most

widely used in applications is the *K-means* clustering (also called *Lloyd's algorithm*, see [36]), fully explained in sect. 3.2.2.1.

1.1.1.4 Distribution-based Clustering

This category defines clusters simply as a set of samples belonging most likely to the same distribution. This statistical and probabilistic point of view gives a great theoretical foundation to this method and allows to create very complex models, which can usually well explain the data structure. Unluckily, they tend to *overfit* the data, i.e. they are not able to generalize what they learn and may therefore fail in assigning a cluster to new samples. This phenomenon could be reduced, for example, decreasing the model complexity.
One of the best performing distribution-based clustering algorithm is the *Expectation-Maximization* (EM) one, which exploits the so-called *Gaussian Mixture Models* (GMMs) and will then be explained more in detail (3.2.1).

1.2 Supervised Learning

Supervised learning is a branch of machine learning that aims to create a function that maps an input into an output based on training examples pairs $(input, label)$ ([41]). It is mainly used to solve two different tasks: *classification* and *regression*. Here we exploit only the first one, which consists in being able to assign a label to new observation based on the characteristics of each class learned using the training set. This training set is employed from the learning algorithm to produce the above mentioned function, which afterwards is used to map, i.e. to classify, the new observations (see Figure 1.4). To correctly classify these new observations, the algorithm has to be able to generalize as much as possible the "patterns" learned from the labeled inputs.

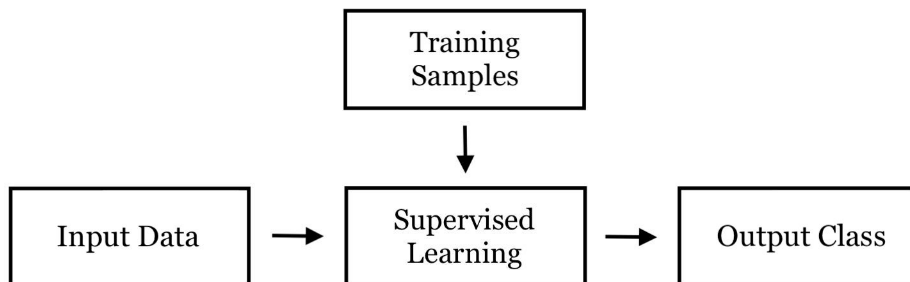


Figure 1.4: Supervised Learning simple block scheme.

Let now $\{(x_1, y_1), ..., (x_I, y_I)\}$ be the *training data*, where the x_i and y_i represent respectively the ith observation and its associated label. The objective of a generic supervised learning algorithm is to search for a function $g : X \rightarrow Y$, with X the samples space and Y the labels one, that should be able to correctly classify the inputs.

The strongest hypothesis of supervised classification theory is that the samples in the training set are independent and identically distributed pairs (i.i.d.). This assumption tends to simplify the underlying mathematics of a lot of statistical learning tasks and, even if it is often violated in real world, is likely to work well in practice problems.
We now introduce the *Loss function* $L : Y \times Y \to \mathbb{R}^+$, which allows to evaluate how good is the function g at explaining the data and intuitively represents the "cost" associated to a mistake of g in classifying samples. Said this, let's define the so-called *expected loss* or *empirical risk* of g:

$$R(g) = \frac{1}{I} \sum_{i=1}^{I} L(y_i, g(x_i)) \tag{1.4}$$

What the learning algorithms tries to do is minimizing, over g, this risk (*empirical risk minimization*), or a slightly different function $J(g) = R(g) + \lambda C(g)$ (*structural risk minimization*), where $C(g)$ represents a penalty function that helps reducing overfitting and regularizes the optimization, while $\lambda > 0$ is a hyperparameter that, in general, is empirically chosen by *cross-validation* and controls the bias-variance tradeoff ([22]). The penalties most commonly applied are the L^2 and the L^1 norms. Depending on the selected learning algorithm, the choice of the loss function may vary a lot.
In the context of supervised learning classification, there exist so many different algorithms - and also variants of the same algorithm - that would be impossible to completely describe all of them here. In this thesis, we just make use of *Support Vector Machine* and a specific case of *Artificial Neural Networks* algorithms, and hence they will be addressed and explained more in details.

1.2.1 Artificial Neural Networks

An *Artificial Neural Network* (ANN), often simply called *Neural Network* (NN), is a computing system that is inspired by the architecture of the human brain to perform tasks like, for instance, image recognition ([63]). It is based on the presence of "units", usually referred as *neurons*, that process and transfer information (signals) between each other through connections (see Figure 1.5). These neurons receive an input and, after processing it, they produce an output. Each input of a neuron is either the output of another neuron or an external feature set of values. The connections are the ANN components that allow to transfer information between neurons. They do that assigning to every neuron output a *weight* representing its importance in the network. The neurons are usually grouped in three different types of layer: input, output or hidden (i.e. internal to the network).
Mathematically, the action of a neuron is the following:

$$y = \psi \left(\sum_i x_i w_i + b \right) \tag{1.5}$$

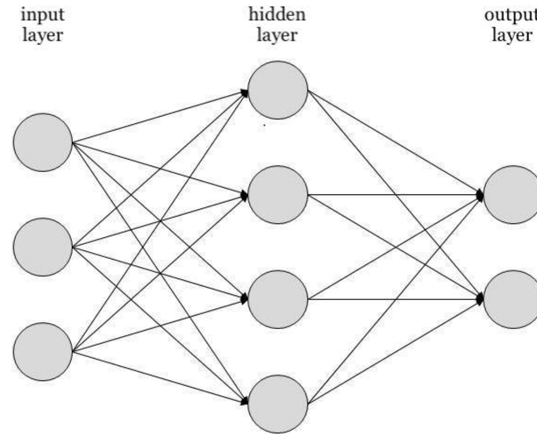


Figure 1.5: Generic structure of a 1-hidden layer ANN with 9 total neurons.

where y is its output, ψ is the *activation function* (see 3.1.2.3), x_i and w_i are respectively the ith input and its weight and b is a coefficient called *bias* which allows to perform a shift of the activation function. In the context of supervised binary classification, the easiest neural structure is the so-called *Perceptron*, first introduced in 1957 by Frank Rosenblatt ([21]). Its structure, shown in Figure 1.6, is very simple: all the nodes in the input layer are directly connected to a single output node (i.e. there is no hidden layer) that uses as activation a *Heaviside* function or a shifting thereof (Figure 3.4). In formula, setting t as the threshold:

$$\psi(x) = \begin{cases} 1 & x \geq t \\ 0 & x < t \end{cases} \tag{1.6}$$

This structure is very straightforward and does not allow to solve very difficult tasks. In the last decades, many improvements were introduced to this simple framework in order to face all kind of machine learning problems. The two main types of ANNs are the *Recurrent Neural Networks* (RNNs) and the *Feedforward Neural Networks* (FNNs) ([28]). Even if in this work we use just a specific subfamily of the second type, also the first one is briefly introduced for completeness.

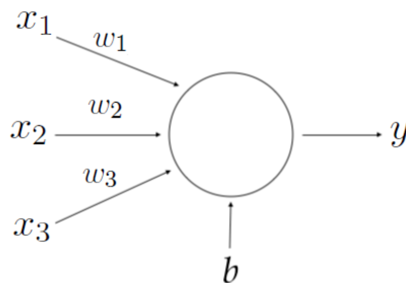


Figure 1.6: Structure of a perceptron with 3 inputs and a bias. x_i are the inputs, w_i their weights, b is the bias and y is its output.

1.2.1.1 Feedforward Neural Networks

This family of NNs is the straightforward evolution of the concept of *perceptron*. They are based on the fact that connections do not form cycles and the information starts from the input layer and can only move forward in the network. These networks can present many hidden layers and can be divided in subfamilies depending, for instance, on the activation they use or on their specific architecture. We list some of them (other than perceptron) and give a brief description of their differences:

- *Multi-layer Perceptron*: this type of FNN is the first generalization of the perceptron and consists in the union of the neurons to form many layers. The information is transmitted from the input layer to the output one creating connections between them and the hidden layers. In the general case, the layers are *fully connected*, which means that every neuron belonging to a layer receives as input the weighted average of the outputs of all the neurons in the previous layer. This means that every neuron output is passed to all the neurons of the following layer. The Figure 1.5 already showed this structure.

- *Radial Basis Function Network*: it is a variant of the more general multi-layer perceptron, in which the activation function is the so-called *radial basic function* and not the one shown in (1.6). This is a real-valued function ψ that depends only on the distance between a point c and the input x, i.e. $\psi(x) = \psi(\,||x - c||\,)$, where $||\cdot||$ is a chosen norm and c could be the origin or another fixed point in the space of the inputs. In general, this network has just one hidden layer and the chosen norm is the Euclidian, even if it has been proven that also the Mahalanobis one perform well in image recognition problems. Usually the taken RBF is the Gaussian:
$$\psi(x - c) = e^{-\epsilon\,||x-c||^2} \tag{1.7}$$
with ϵ tuning parameter. This network often performs very well in *function approximation* problems.

- *Convolutional Neural Networks*: these networks differ from the standard FNNs because their hidden layers are not fully connected. In particular, they exploit an operation called *convolution* which allows to detect the approximate location of important and specific patterns and features in the inputs. They are very used in the field of image classification and recognition and that's why we are going to describe them in details in sect. 3.1, where they are used to extract highly specific features from images.

Thanks to their forward structure, the most suitable and widely used learning technique for FNNs is the so-called *backpropagation* algorithm.

1.2.1.2 Recurrent Neural Networks

This type of ANNs has a structure that usually presents cycles (also called loops) that connect the layers between each other and allow to transfer the information both forward and backward during the training step, giving a dynamic temporal behaviour to the network. In particular, each of the neurons inside the hidden layers receives the input with a specific delay in time. This characteristic makes the RNNs particularly suitable in solving tasks like *time series prediction and anomalies detection* ([13]) and *speech synthesis and recognition* ([64]).
Since there exist several different types of RNNs and they are introduced here only for completeness, we refer to [39] for a more detailed description.

1.2.1.3 The Backpropagation Algorithm

The *backpropagation* algorithm is used to make a feedforward neural network (1.2.1.1) learn to detect the input characteristics. Its purpose is to efficiently pass the learned information to all its layers through an updating of their parameters. To explain how it works we first introduce some notations.
Let O_n and W_n be respectively the output and the set of parameters of the nth layer of the network, with $n = 1, ..., N$ (W_n and O_n are actually subsets of the network parameters and outputs sets W and O). Let $O_0 = X = \{x_1, ..., x_I\}$ be the set of inputs of the first layer. From the general structure of a FNN, we can write $O_n = F(W_n, O_{n-1})$, where F represents the chosen activation function (3.1.2.3). We also call L_i the value of the loss/cost function given the input x_i.
As explained in [33], assuming we know the partial derivative of L_i with respect to any layer output O_n, it is possible to use the backward recurrence and write:

$$\frac{\partial L_i}{\partial W_n} = \frac{\partial F}{\partial W}(W_n, O_{n-1}) \frac{\partial L_i}{\partial O_n} \tag{1.8}$$

$$\frac{\partial L_i}{\partial O_{n-1}} = \frac{\partial F}{\partial O}(W_n, O_{n-1}) \frac{\partial L_i}{\partial O_n} \tag{1.9}$$

Here, $\frac{\partial F}{\partial W}(W_n, O_{n-1})$ and $\frac{\partial F}{\partial O}(W_n, O_{n-1})$ are respectively the *Jacobian* matrices of F with respect to the weights W and the outputs O, evaluated in the point (W_n, O_{n-1}). When these equations are applied in reverse to the network layers, i.e. from the Nth to the 1st one, all the derivatives of the loss function with respect to weights can be calculated. Finding the gradient of the loss function "layer by layer" is computationally much cheaper than doing that for each weight individually. This is why the backpropagation algorithm is so efficient and hence very commonly used.
The easiest learning procedure to adjust the weights, in order to minimize the loss function in the backpropagation setting, is the so-called *gradient descent*.

1.2.1.4 Gradient Descent

The *gradient descent* is one of the most well performing optimization algorithms and undoubtedly the most common in the field of neural networks. It consists of an iterative process that aims to find a local minimum of a certain objective function. From now on we use the same notation adopted in the backpropagation section.
Given a function $L(W)$ that depends on the variable W and a point $\hat{W}$ in the input space of L, we know that if the gradient of the function evaluated in that point, i.e. $\frac{\partial L}{\partial W}(\hat{W})$, is non-zero, the direction of the gradient coincides with the direction of maximum growth of the function L from point $\hat{W}$. This is a useful property of differentiable functions which is used in optimization problems to minimize a generic loss function and, hence, to efficiently update the parameters of the network during training. In particular, when trying to minimize the loss, the parameters are updated moving in the direction of the negative of the gradient. If we instead move in the direction of the positive of the gradient, the process reaches a local maximum of L and it is called *gradient ascent*. A specific variant of the gradient ascent is the *Stochastic Dual Coordinate Ascent* (SDCA) and it is used to solve, for instance, the maximization of the SVM dual problem ([54]).
Sticking to the case of a minimization problem, the formula for the parameters updating between iteration t and $t+1$ is the following:

$$W_{t+1} = W_t - \eta \frac{\partial L}{\partial W}(W_t) \tag{1.10}$$

where $\eta \in \mathbb{R}^+$ is a small positive real number that defines the step size and is called *learning rate*. Its value could be fixed a priori or vary during the updating process.
In general the algorithm guarantees convergence just to a local minimum, but if the objective function L is convex the convergence is global. This is why the initialization of the parameters is usually not negligible. To reach convergence, this method could take many iterations since, often, the closer to the minimum, the smaller the value of the gradient, i.e. the step toward the searched minimum. Check Figure 1.7 to have a simple visual representation of the algorithm. There exist three variants of the gradient descent, which differ in terms of the amount of data used to compute the gradient of the objective function needed to adjust the network weights. This allows to make a trade-off between the time needed to find the parameters update and its accuracy with respect to the descending direction ([50]).

Batch Gradient Descent

The *batch gradient descent*, also called *Vanilla gradient descent*, is the basic version of the gradient descent algorithm. It computes the gradient of the objective function using all the available training data at once and hence performing just one update for each of the preset epochs, which are the number of passes of the entire training dataset

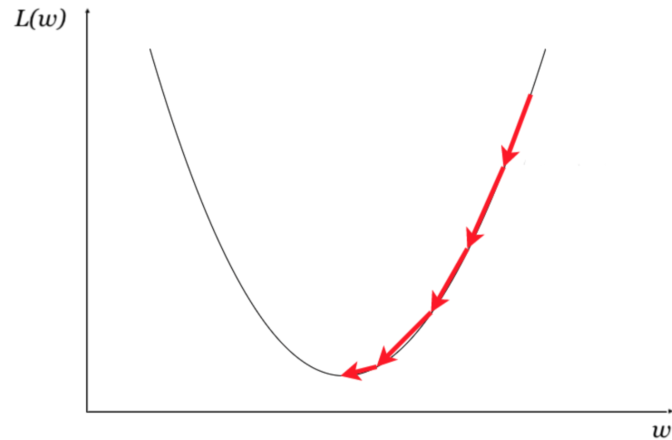


Figure 1.7: $L : \mathbb{R} \rightarrow \mathbb{R}$ is a simple convex function. Here each red arrow represents the direction of the negative gradient that determines the parameters updating. Note how it gets shorter and shorter approaching the minimum.

done by the machine learning algorithm:

$$W_{fin} = W_{in} - \eta \frac{\partial L}{\partial W}(W_{in}) \tag{1.11}$$

This variant always reaches at least a local minimum, but has the disadvantages of taking a very long time to find the result and being infeasible with large training sets that do not fit in memory all at once. Morover, it recomputes gradients for similar examples before each parameter update (redundant information) and finally it prevents an online actualization of the parameters with new observations.

Stochastic Gradient Descent

Stochastic gradient descent (SGD), unlike the already described batch gradient descent, computes the parameters update for each training example in the data. In formula the update for the $(t+1)^{th}$ iteration:

$$W_{t+1} = W_t - \eta \frac{\partial L}{\partial W}(W_t\,;(x_i, y_i)) \tag{1.12}$$

where x_i is the ith training example and y_i its label. An important thing to do when applying SGD is shuffling data at each epoch, since the order of the inputs may bias the optimization.
The power of this method is that it eliminates redundancy of information and allows online actualization. As a consequence of the one-sample updating, the algorithm results very fast, but it usually presents *fluctuations*, which could complicate the convergence to the exact minimum during training due to the fact that the gradient direction may

not be the best one. In practice, setting a decreasing learning rate with respect to the epochs could help with this problem: it makes the SDG converge almost surely to a local or a global minimum ([30]).

Mini-batch Gradient Descent

The *mini-batch gradient descent* tries to join both the already presented methods, performing an update for every *mini-batch* of n training samples:

$$W_{t+1} = W_t - \frac{\eta}{n} \sum_{i=1}^{n} \frac{\partial L}{\partial W}(W_t\,; (x_i, y_i)) \tag{1.13}$$

where the x_i are the n training samples inside the considered batch and the y_i their labels. Note that to update the weights, this method uses the average of all the gradients evaluated on all the inputs inside the considered batch. The commonly used number of samples inside the batches ranges between 50 and 256, but can vary depending on the applications ([50]). The trade-off highlighted by the batch size is the following: a small batch makes the parameters updating very fast, but sometimes it causes the objective function to diverge; a big batch size tends to eliminate fluctuations, but drastically slows down the algorithm. For how it was built, this approach embraces the main advantages of both the batch and stochastic gradient descents and deals very well with their weaknesses. This makes it the best performing, and hence the most widely used in many optimization problems, especially for neural networks learning tasks.
One of the things about gradient descent that really affects its convergence, besides the batch size and the vanishing gradient problem (3.1.2.3), it is the value chosen for the learning rates. For completeness, we just cite two techniques to do that in an efficient way even if they are not being used in this work: via *line search* or using the *Barzilai–Borwein* method.

1.2.2 Support Vector Machine

In this section is explained the theory behind a *Support Vector Machine* (SVM) classifier and what its advantages are in applications. As already said, in supervised classification problems it is called *training data* the set of all the given points $\{(\mathbf{x}_i, y_i)\}_{i=1}^{I}$, where $\mathbf{x}_i$ is the feature vector that describes the data point, y_i takes the value ± 1 and is the label of the ith point[1], I is the number of samples in the set. SVM training algorithm builds a model that assigns new examples to one class or the other, making it a non-probabilistic classifier. This classifier that SVM aims to construct is of the form:

$$y(\mathbf{x}) = sign[f(\mathbf{x})] \tag{1.14}$$

[1] The SVM was originally invented for *binary* classification problems and all the theory is based on this hypothesis. The multi-class adaptation is going to be explained in a few pages.

where $f(\mathbf{x})$ is a *decision rule* function that changes depending on the complexity of the problem.

Hard-Margin SVM

This subsection is facing the special case of *linearly separable data*. Suppose to know that there exists a surface in n dimensions that completely separates the training data, i.e. an $n-1$ dimensional hyperplane defined by:

$$f(\mathbf{x}) \equiv \mathbf{w} \cdot \mathbf{x} + b = 0. \tag{1.15}$$

Here $\mathbf{w}$ represents the normal vector to the hyperplane and b its offset from the origin. As shown in Figure 1.8, since the data is linearly separable, we can find the two parallel hyperplanes that separate the data and such that the distance between them is as large as possible (they are called *support vectors*). The region between them is called *margin* and the maximum-margin (or largest margin) hyperplane is the one that lies halfway between them.

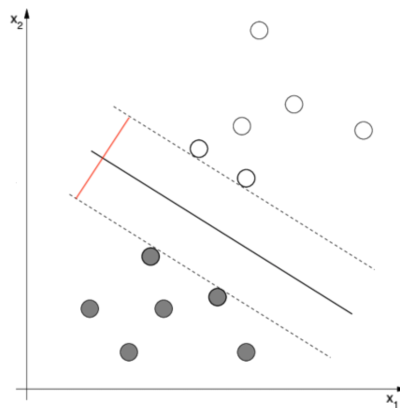


Figure 1.8: 2-dimenensional example of a SVM classifier for *linearly separable data*. The red line represents the margin, the black line is the classification hyperplane, i.e. $f(\mathbf{x}) = 0$, while the dashed ones are the support vectors, i.e. $f(\mathbf{x}) = \pm 1$.

The regions defined by the support vectors are:

$$\begin{cases} \mathbf{w} \cdot \mathbf{x}_i + b \geq +1 & y_i = +1 \\ \mathbf{w} \cdot \mathbf{x}_i + b \leq -1 & y_i = -1 \end{cases} \tag{1.16}$$

which could be rewritten as:

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 \tag{1.17}$$

Geometrically speaking, the distance between them is $\frac{2}{||\mathbf{w}||_2}$. What the SVM algorithm tries to do is maximizing it, i.e. minimizing $||\mathbf{w}||_2$, applying the constraint showed in

(1.17). The resulting problem is then:

$$\boxed{\begin{array}{lll} \underset{\mathbf{w}}{\text{minimize:}} & \frac{1}{2}\mathbf{w}\cdot\mathbf{w} & \\ \text{subject to:} & y_i(\mathbf{w}\cdot\mathbf{x}_i + b) \geq 1 & i = 1, ..., I \end{array}} \tag{1.18}$$

The factor $\frac{1}{2}$ in the objective function is just a coefficient that helps simplify some algebra. The important consequence of this description is that the largest margin hyperplane is found just by looking at those $\mathbf{x}_i$'s that lie next to it and not all the data.

Soft-Margin SVM

Supposing that the data is *linearly separable* actually represents an unrealistic assumption. To generalize the problem, this hypothesis is removed and in each constraint we introduce a new *slack variable* ξ_i that acts as follows:

$$y_i(\mathbf{w}\cdot\mathbf{x}_i + b) \geq 1 - \xi_i \tag{1.19}$$

with $\xi_i \geq 0$. The new problem is then the following[2]:

$$\boxed{\begin{array}{lll} \underset{\mathbf{w}}{\text{minimize:}} & \frac{1}{2}\mathbf{w}\cdot\mathbf{w} + \lambda\sum_i \xi_i & \\ \text{subject to:} & \xi_i \geq 0, & \\ & y_i(\mathbf{w}\cdot\mathbf{x}_i + b) \geq 1 - \xi_i & i = 1, ..., I \end{array}} \tag{1.20}$$

In practice, ξ_i is usually calculated using the so-called *Hinge Loss* function as:

$$\xi_i = \max(0, 1 - y_i(\mathbf{w}\cdot\mathbf{x}_i + b)) \tag{1.21}$$

Note that $\xi_i = 0$ when (1.17) is satisfied, i.e. when the point is correctly classified using the hyperplane found with *Hard-Margin SVM*, while if $\xi_i > 0$ the point cannot be linearly separated. Here ξ_i represents the amount of discrepancy in the classification of $\mathbf{x}_i$, i.e. the distance of $\mathbf{x}_i$ from the corresponding class margin. $0 < \lambda < \infty$ represents a *trade-off* hyperparameter and its value could be improved with a process called *regularization*. After all these considerations we could finally say that the *Soft-Margin SVM* is a generalization of the *Hard-Margin SVM* specific case.

1.2.2.1 Computation of the SVM Classifier

In this section is explained how the SVM is computed, starting from defining another way to see the minimization problem and then describing the so-called *kernel trick*.

[2] One possible variant is to use the *Least Squares* function as regularization term, i.e. $\sum_i \xi_i^2$, but since it leads to more complicated equations and it has not been used in this thesis, it is not taken into account.

Primal and Dual Problems

The problems in (1.18) and (1.20) are described as *quadratic programming* problems, i.e. optimization problems with a quadratic objective function in at least one variable (in our case $\mathbf{w}$), subject to linear constraints over that same variable. These problems are called *Primal Problems* of the two SVMs described algorithms. Every primal problem has a *Dual Problem*, which could be easily thought of a different way to solve the primal one. In a nutshell, the process consists in finding the *Lagrangian* of the primal problem:

$$\mathcal{L} = \frac{1}{2}\mathbf{w} \cdot \mathbf{w} + \sum_i \alpha_i [1 - y_i(\mathbf{w} \cdot \mathbf{x}_i + b)] \tag{1.22}$$

which is the objective function incorporating the constraints. α_i is the ith component of the *Lagrange multiplier vector* and represents the new objective variable. The following extremum conditions:

$$0 = \frac{\partial \mathcal{L}}{\partial \mathbf{w}} = \mathbf{w} - \sum_i \alpha_i y_i \mathbf{x}_i \tag{1.23}$$

$$0 = \frac{\partial \mathcal{L}}{\partial b} = \sum_i \alpha_i y_i \tag{1.24}$$

are then substituted in (1.22). After some calculation and rearranging the result, we obtain the following dual problem:

$$\boxed{\begin{array}{ll} \underset{\boldsymbol{\alpha}}{\text{minimize:}} & \frac{1}{2}\sum_{i,j} \alpha_i y_i (\mathbf{x}_i \cdot \mathbf{x}_j) y_j \alpha_j \; - \sum_i \alpha_i \\ \text{subject to:} & \boldsymbol{\alpha} \cdot \mathbf{y} = 0, \\ & 0 \leq \alpha_i \leq \lambda \qquad i = 1, ..., N \end{array}} \tag{1.25}$$

The important result, which follows from *Strong Duality* and *Kuhn-Tucker* theorems (to have more details see respectively [14] and [25]), is that the solution from the dual problem $\hat{\boldsymbol{\alpha}}$ coincides with the solution of the primal $\hat{\mathbf{w}}$ and $\hat{b}$. Once $\hat{\boldsymbol{\alpha}}$ is found, we just need to substitute its value in (1.23) to find $\hat{\mathbf{w}}$ and then to substitute $\hat{\mathbf{w}}$ in the first of the following *Karush-Kuhn-Tucker* conditions to find $\hat{b}$:

$$\hat{\alpha}_i \left[y_i(\hat{\mathbf{w}} \cdot \mathbf{x}_i + \hat{b}) - 1 + \hat{\xi}_i \right] = 0 \tag{1.26}$$

$$(\hat{\alpha}_i - \lambda)\hat{\xi}_i = 0 \tag{1.27}$$

It is now straight forward that:

- $\hat{\alpha}_i = 0$: the data point $\mathbf{x}_i$ is on the correct side of the margin.
- $0 < \hat{\alpha}_i < \lambda$: the data point $\mathbf{x}_i$ lies exactly on a support vector.
- $\hat{\alpha}_i = \lambda$: the data point $\mathbf{x}_i$ is inside or on the wrong side of the margin.

What makes the usage of the dual so helpful is that almost all the calculations performed to obtain its solution scale with the number of data points I (just the dot product in the objective function scales with the dimensionality of the feature vector). Even if it might seems odd in problems with many data points, in many others, as for example *image processing* with not a lot of data but with huge feature vectors, this could lead to a speed up of the training process (see SDCA in 1.2.1.4).

Kernel Trick

Suppose now that we would like to map a d-dimensional feature vector $\mathbf{x}$ into a D-dimensional space, with $D > d$. This could be done imaging an embedding function ψ as shown in the Figure 1.9.

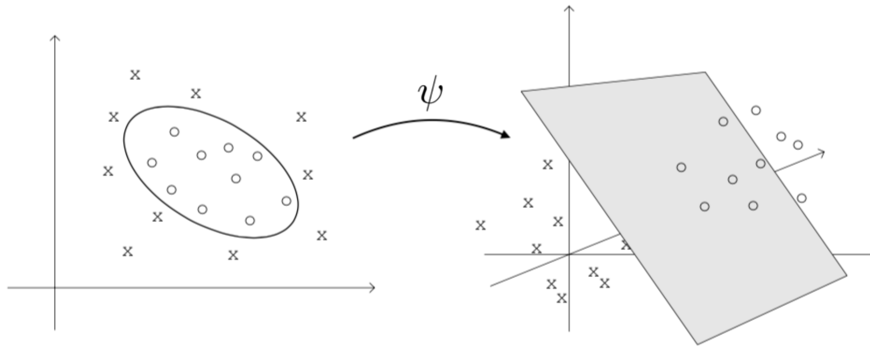


Figure 1.9: Example of observations mapped from a 2D space to a 3D one, where non-linear separation surfaces can become well approximated by linear ones. In real world applications, very high dimensional embedding spaces are used.

The fundamental concept behind this function is that in D-dimensional space it is possible to linear classify some data which, in the original space, could be labeled only with a non-linear separating surface. Moreover, thanks to the dual formulation of the problem, the dimension D could be set to be very high (i.e. a more precise linear classifier) without changing the complexity of the problem, since as explained in the previous section, it depends only on the number of data points.
We then introduce the so-called *kernel function*:

$$k_{i,j} \equiv k(\mathbf{x}_i, \mathbf{x}_j) \equiv \psi(\mathbf{x}_i) \cdot \psi(\mathbf{x}_j) \tag{1.28}$$

which has the properties of being symmetric and having non-negative eigenvalues. Setting this kernel in equation (1.25) in place of the dot product $\mathbf{x}_i \cdot \mathbf{x}_j$ and after some calculation, it is easy to find the classification vector $\hat{\mathbf{w}}$ in the transformed space:

$$\hat{\mathbf{w}} = \sum_i \hat{\alpha}_i y_i \psi(\mathbf{x}_i) \tag{1.29}$$

The bias $\hat{b}$ is calculated as before substituting in the *Karush-Kuhn-Tucker* equations. Note that we do not actually need to know the explicit expression of the function ψ, but only the matrix formulation of the kernel $k_{i,j}$. Some example of good working kernels are[3]:

$$\begin{aligned}
\text{Linear:} \quad & k(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i \cdot \mathbf{x}_j \\
\text{Power:} \quad & k(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j)^p \\
\text{Polynomial:} \quad & k(\mathbf{x}_i, \mathbf{x}_j) = (a\mathbf{x}_i \cdot \mathbf{x}_j + b)^p \\
\text{Sigmoid:} \quad & k(\mathbf{x}_i, \mathbf{x}_j) = \tanh(a\mathbf{x}_i \cdot \mathbf{x}_j + b) \\
\text{Gaussian Radial Basis function:} \quad & k(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(- \frac{|\mathbf{x}_i - \mathbf{x}_j|^2}{2\sigma^2} \right)
\end{aligned} \tag{1.30}$$

To conclude, a problem that arises when working in a high dimensional space is that the SVM tends to overfit the data and hence the *generalization error* increases. In practice, given enough data samples, this issue is solved and the algorithm still can achieve good classification performances.

1.2.2.2 Multiclass SVM

The goal of *Multiclass SVM* is to predict labels of samples by reducing the single *multiclass* problem into multiple *binary* problems. There exist many different techniques to do that, like *Directed Acyclic Graph SVM* (or *DAGSVM*) and *Error-Correcting Output Codes*, but the most common and the one we choose to apply in this thesis is the so-called *One-vs-All* one. This method consists in building a binary classifiers that distinguishes between one of the labels and all the rest, and repeating the process for all the existing labels. To apply this method, since the SVM is a deterministic classifier, it is necessary to change the way the class is assigned. In particular, the equation (1.14) is not used anymore and, each time we repeat the binary classification process for one class versus the rest, a transformation of the decision rule function $f(\mathbf{x}_i) = \mathbf{w} \cdot \mathbf{x}_i + b$ is performed and a score is assigned to each input $\mathbf{x}_i$. This process is called *calibration* and can be applied following different criteria. The one used in this thesis works as follows:

$$y_k(\mathbf{x}_i) = \frac{\mathbf{w}}{m_k - m_{rest}} \cdot \mathbf{x}_i + \frac{b - m_{rest}}{m_k - m_{rest}} \tag{1.31}$$

where $i = 1, ..., N$ is the number of inputs, $k = 1, ..., K$ is the number of classes, i.e. the binary classifiers to be trained and m_k and m_{rest} are the medians of the $f(\mathbf{x}_i)$, where the $\mathbf{x}_i$'s belong respectively to class k and to all the rest of the classes. This method could be seen as a sort of "median normalization" of the weights and bias trained with the SVM and that's why the scores $y_k(\mathbf{x}_i)$ are in general very close to zero. This

[3] In this thesis just the linear one was considered, the other proposed kernels could be part of a further work study.

technique is not computationally expensive and, indeed, it is called *cheap calibration*. One of the most common used calibration technique is the *Platt scaling* ([46]), which is more computationally expansive and is based on transforming the outputs of the SVM classifier into a probability distribution over classes. The use of this type of calibration could be implemented in some further works. Regardless of the chosen calibration method, the classification of new instances is performed by using a "*winner-takes-all*" strategy: the classifier with the function that leads to the highest output $y_k(\mathbf{x}_i)$, for a given input $\mathbf{x}_i$, is chosen to assign the new instance class.

1.2.3 Problem of Imbalanced Classes

Tackling the problem of binary classification, it happens that the given data is divided in classes which are not balanced, i.e. contain a very different number of samples. In this situation the classifier tends to misclassify the instances of the class containing a low amount of data. The main issue here is that the *accuracy* of the model (i.e the portion of correctly classified samples over the total number of samples) could be high even if the classifier fails in assigning the label to all the observations of a low represented class, just because the misclassified samples are not so much. The *imbalance problem* appears in many real data analysis, but in general for only two different reasons. The first one is that it is intrinsic to the real problem, e.g. in fraud/terrorist detection analysis, where usually the number of fraud observations is much smaller than the "honest" one ([43]). The second cause is simply due to how the samples are collected and it does not depend on the extent of the problem, like it happens in the dataset used for this work (see section 5.1 for a precise description of the used dataset).
To solve this problem there exist many different approaches. Nowadays, *Precision-Recall* (PR) *Curve* and *Receiver Operating Characteristic* (ROC) *Curve* are two of the best and most widely used. The majority of the studies that deal with imbalanced data employ the ROC approach, but we choose to apply the PR one because it leads to a more accurate and intuitive interpretation of practical classifier performance ([51]).

1.2.3.1 Precision-Recall Curve

The *Precision-Recall* (PR) *Curve* is a technique that allows to evaluate the performance of a binary classifier trained on imbalanced classes. As already said, the accuracy, calculated as:

$$acc = \frac{TP + TN}{TP + FP + TN + FN} \tag{1.32}$$

could fail in detecting misclassification for low represented classes. To deal with this problem, in the PR method are introduced the following two quantities:

- Precision of class C:

$$p(C) = \frac{TP}{TP + FP} \tag{1.33}$$

- Recall of class C:

$$r(C) = \frac{TP}{TP + FN} \tag{1.34}$$

Check the Figure 1.10 to see the definition of the used variables and an example of a generic *confusion matrix* for a binary classification problem.

		Ground-truth label	
		+	-
Predicted label	+	TP	FP
	-	FN	TN

Figure 1.10: Confusion matrix of a generic binary classification problem. "+" and "-" represent the positive and negative class. *TP* and *TN* are respectively the number of positive and negative samples correctly classified, while *FP* and *FN* represent instead the negative and positive misclassified observations.

For a given class C, a precision score of 1 tells that all the samples predicted as belonging to class C are correctly classified, while a recall score of 1 means that every sample that belongs to class C is correctly classified. Since they give two different interpretations of the goodness of the performance and between them usually exist an inverse relation, they are both used to evaluate a classifier. Sometimes they are also combined to form the so-called F_1*-score*, defined as follow:

$$F_1(C) = 2 \times \frac{p(C)\, r(C)}{p(C) + r(C)} \tag{1.35}$$

Besides that, the thing in which we focus is the plot of these two quantities, which allows the visualization of performance. This curve is constructed by first plotting precision-recall pairs, or points, that are obtained using different thresholds on a probabilistic or other continuous-output classifier ([15]). A generic example of PR curve is shown in Figure 1.11.

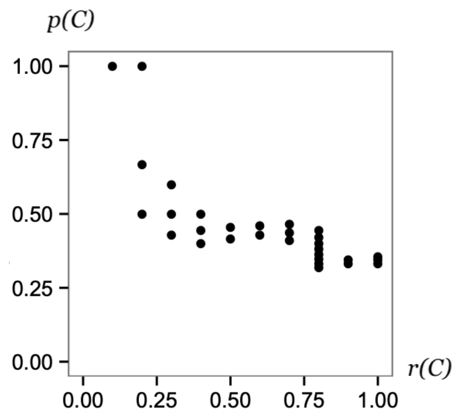


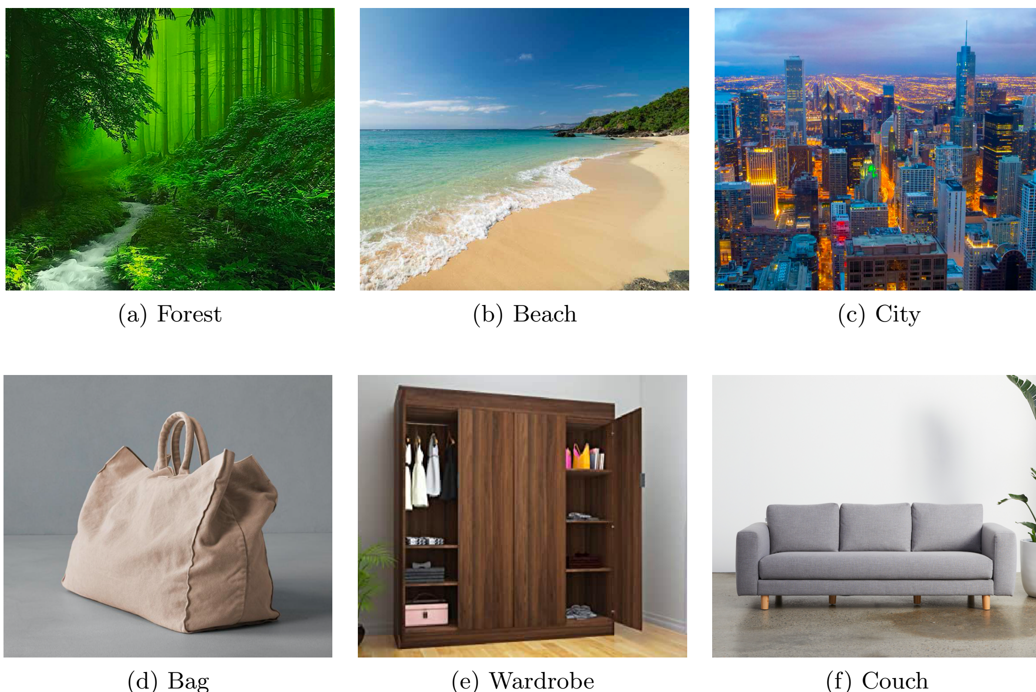
Figure 1.11: Simple example of a Precision-Recall curve with 10 positive samples (i.e. in C) and 20 negative ones. Taken from [15].

What is really important about this plot is the *area under the PR curve* (AUCPR), also called *average precision* (AP), which is a single number summary of the information given by the curve. This quantity represents a score that can be used as comparison between different models and, since both precision and recall lie in $[0, 1]$, the value of the area also lies in this interval. In particular, the higher the AP, the better the model capability of identifying, and hence classifying, the samples of a certain class C. Since the average precision is calculated for each given class, to evaluate the total model performance it is introduced the *mean average precision* (mAP), which is nothing more than the average of the APs of all the classes in the dataset.

Chapter 2

Image Classification Problem

As already explained in section 1.2, *classification* is a supervised machine learning task that arises when, given an already labeled set of data, it is necessary or simply can be useful to identify the common patterns shared by the observations in the training set that belong to the same class. Since this task appeared to be very common in a lot of research fields, a big slice of the scientific community have contributed, especially in the last decades, to develop advanced classification approaches for improving the classification accuracy.



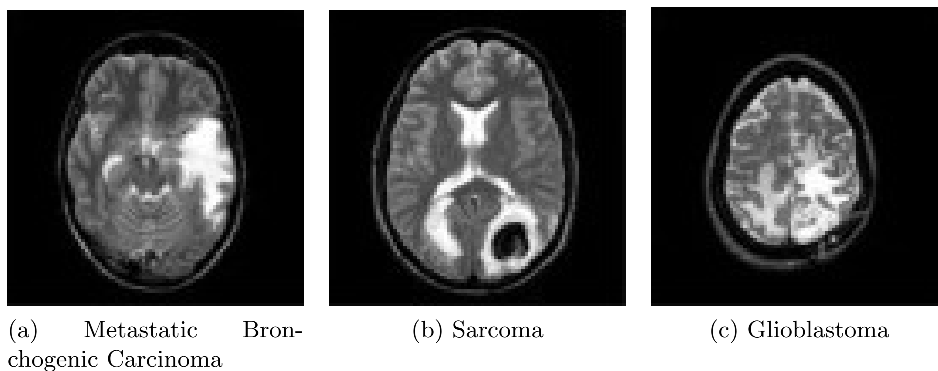
(a) Forest (b) Beach (c) City

(d) Bag (e) Wardrobe (f) Couch

Figure 2.1: Possible generic images inside a dataset. 2.1a, 2.1b and 2.1c represent examples of *Scene* images, while 2.1d, 2.1e and 2.1f are instead examples of *Object* images.

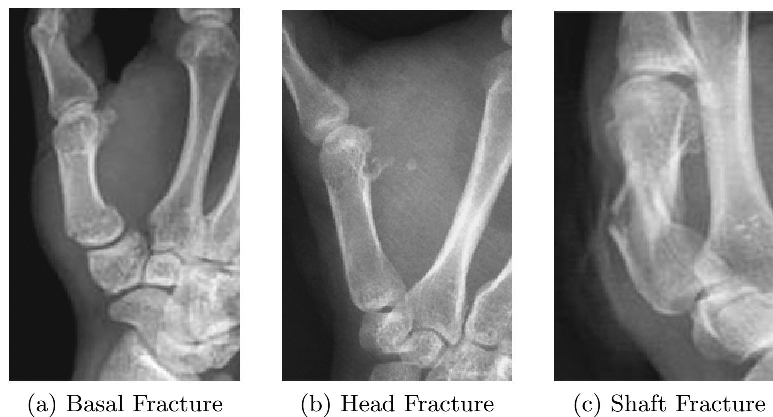
For what concerns *image classification*, the given data consists in a set of images (see Figure 2.1) that could be divided into *scene* and *object* ones. In this context, the classification algorithms tries to learn the *features* that a certain class of images can present and, by checking their presence, to decide if the selected image belongs or not to a certain group. To perform classification, it is better to pass from image features instead of directly use the original images. This because the features represent, loosely speaking, salient points on the image and should be invariant to image transformations like rotation, translation, scaling, lighting conditions and color, even if not without a limit. In practice, this represents a complex problem and a challenge in the machine learning research field: solving this task with an high accuracy requires an algorithm that is able to efficiently identify the important features in the input images. Usually, a great amount of samples in the dataset helps the learning of these features, but this is something that in real-world is not always feasible.

2.1 Medical Image Classification



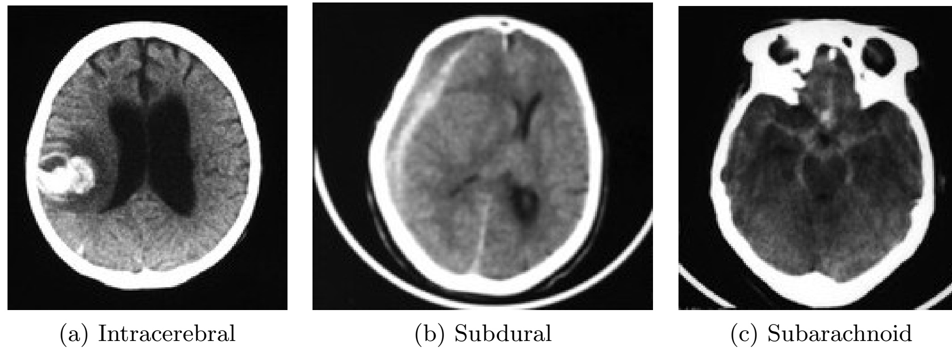
(a) Metastatic Bronchogenic Carcinoma (b) Sarcoma (c) Glioblastoma

Figure 2.2: Brain Tumor Images. Taken from [42].



(a) Basal Fracture (b) Head Fracture (c) Shaft Fracture

Figure 2.3: Hands and Wrist Bone Fractures Images. Taken from [66].



(a) Intracerebral (b) Subdural (c) Subarachnoid

Figure 2.4: Intracranial Hemorrhages Images. Taken from [26].

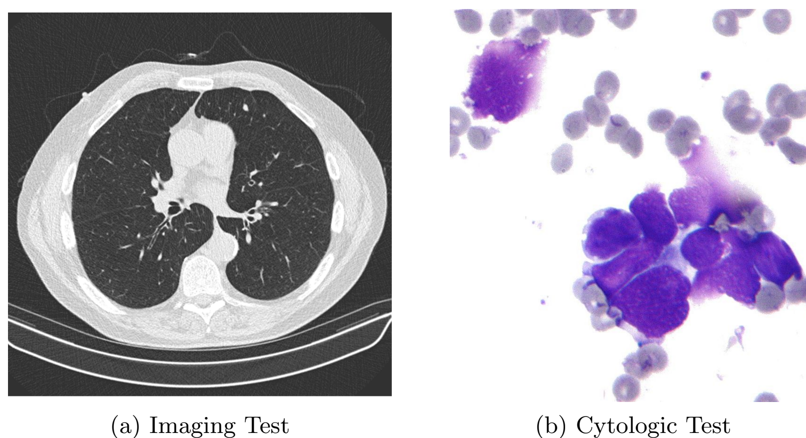
In image recognition area, *medical image classification* proved to be a very useful tool, helping doctors in disease diagnosis and stimulating many further researches. Its aim is to classify medical images into different categories, which is a task that can overall be divided in two steps: the extraction of images important features and the creation of a model able to interpret those features in order to classify the input images.
Nowadays, the diseases in which this process is efficiently applied range from tumor detection (Figure 2.2), bone fracture detection (Figure 2.3), identification of intracranial hemorrhage (Figure 2.5) and many others. In the world of *tumor detection*, one of the most important steps to be done in order to perform classification concerns the way images are collected before being analyzed. We present briefly some of these techniques for the specific type of cancer we aim to correctly classify: the *lung cancer*.

2.1.1 Lung Cancer Detection

In the field of medical research there exist many different tests that can be asked to be done in order to check the presence of lung cancer. They differ in terms of price, process time, accuracy in detecting the tumor and some of them result much more invasive and harmful for the human body than others. Some of these techniques can be grouped in two principal categories: *imaging tests* and *cytologic tests*.

- *Imaging test*: the aim of this type of tests is to identify the presence of an abnormal mass or nodule inside the lungs looking at the inside of the body. This could be done in many ways. The first is *X-ray*, which is not so accurate and is usually followed by a *Chest* scan (also called CT scan), since it is very good in examining lung tissue. The second one is the so-called *Magnetic Resonance Imaging* (MRI) scan, which gives better results when the images need to be very detailed, looking for cancer at places where, for instance, bone might interfere. The first method is usually cheaper, but expose the patient to a minimal radiation that in some situations, like pregnancy, is not appropriate. The test time also differs a lot, for CT scan it takes approximately 5 minutes, while for MRI scan it could take from 15 minutes to 2 hours, depending on the part of the body being examinated.

- *Cytologic test*: in this category of tests, differently from before, the information is obtained by examining the structure of specific cells. The process of studying cells, group of cells and tissues is called *Histological* analysis and it uses techniques like microscopic imaging technology and stains to detect the microscopic changes occurring at cellular and tissue level. The first cytoligic method we describe is the *Sputum* cytology, which looks for cancerous cells in lung secretions or phlegm. The patient coughs up a sample of mucus, which is viewed under the microscope to check the tumor presence. The other way to obtain these cells is much more common and is performed by *Biopsy*. In this case, the extraction of the suspicious tissue cells can be done passing through the main airways of the lungs (*Transbronchial* biopsy) or through the chest (*Needle*, *Thoracoscopic* and *Open* biopsy). Some of these techniques are very invasive, need general anesthesia and are requested just if no other method was able to detect the tumor presence and type. They are not so expansive, but sometimes cannot be performed without a detailed map of the lungs (obtained for instance with CT scan). Their advantages are that they give a very detailed information about the tumor type (which is something that imaging tests do not always reach) and that the whole procedure usually takes 15 minutes maximum (10-20 seconds for each cell sample).



(a) Imaging Test (b) Cytologic Test

Figure 2.5: 2.5a: imaging test via CT scan. 2.5b cytologic test via biopsy.

2.2 Problem Formulation

The innovation that machine learning brought in the field of medical image classification is very remarkable. Every year, cutting edge techniques are developed and more and more efficient methods are introduced and applied to solve new tasks.
In this work, we use cell images that are obtained digitalizing brush cytologic preparations, as it will be better explained in section 5.1. We believe that this will help in perform a more efficient classification of the lung cancer cells since, in general, images obtained with this procedure are more detailed than the ones obtained by imaging test.

Classifying lung cancer cells is a task that can be efficiently solved, once the images are obtained, by a skilled pathologist. In particular, the process consists in identifying specific anomalies that lung cells could present. In our specific case, as better explained in section 5.1, we have four different classes: adenocarcinoma, epidermoid carcinoma, OAT-cells and negative cells (the ones that haven't been diagnosed with cancer presence). The machine learning classification process is not able to underline the chemical differences between these four categories, like the fact they may or may not react to some chemical agent, so, to describe their characteristics, we stick to their visual traits. The cells affected by adenocarcinoma or epidermoid carcinoma are the most similar and demonstrate poorly differentiated morphological features: they are, in general, medium- or large-sized lung cells with high-grade nuclear atypia and the occasional presence of multinucleous; they also show some gland-like formations and squamous differentiation, respectively ([62]). On the contrary, OAT-cells present more distinct characteristic that lead to a easier identification: small size, a round-to-fusiform shape, scant cytoplasm, finely granular nuclear chromatin and inconspicuous, or even absent, nucleoli ([59]). These morphological attributes are the features that a pathologist look for when asked to classify a lung cell images. In the same way, the aim of a supervised machine learning classifier is to implicitly detect those patterns inside the given images and to learn how to assign them to the lung cancer class they actually belong to.
In the more general area of medical image classification, some of the nowadays main challenges are:

- Lack of collected data. As already said, this is something that often happens in real-world medical problems and it is difficult to deal with. It also remarkably influences the goodness of the performance when training a network from scratch.

- Extract important features from images. In an image classification problem, one of the most important tasks is being able to efficiently identify significative patterns inside an image. The accuracy of the classification depends not only on the amount of data available, but also on the ability of an algorithm to identify these patterns and associate them with the respective class.

- Make features informative for a classifier. An expert pathologist can easily understand the characteristic of a disease in an image of a cell or in an x-ray scan, interpret them and give an appropriate diagnosis. A classifier, instead, needs to be fed with suitable features to be able to interpret them and correctly classify an image.

We try to answer to the tasks above in the next subsection, where we present the results obtained in some of the related works that guided us in formulating our proposed solution.

2.2.1 Related Works

The first listed challenge is one of the most difficult task to solve in medical image classification problems. Usually, patient health data is — for good reasons — well protected by patient data laws and the standards differ considerably from country to country, which makes the issue even more complicated. Apart from the ethic reasons, on which we have no possibility to freely intervene, deep learning experts have found practical methods to use the small amount of available data at best. In [40] it is presented how *data augmentation* can improve the classification performance, without physically collecting new data, in three different medical frameworks: skin melanomas diagnosis, histopathological images and breast magnetic resonance imaging scans analysis. Here are compared traditional methods of data augmentation, based on combinations of affine image transformations and color modifications, with other innovative methods, like GANs, which are based on deep learning models. The first approach is still one of the most used to face the issue of lack of data, since it is fast and easy to implement and is successfully applicable to many fields, but it struggles in adding new visual features to training images. The second approach performs slightly better, but has the flaw of being computationally very expansive. Merging the two methods could bring huge potential for improving data-hungry deep learning algorithms, but would still require a high computational cost.

The second listed challenge can be tackled in many different ways and, to be efficiently solved, gave rise to several techniques. In [35] these techniques are applied and used to classify lung image patches with interstitial lung disease (ILD). In particular, they compared the *Local Binary Patterns* (LBP) and the very popular *Scale-Invariant Features Transform* (SIFT) performances, which are known for being feature extractors that mainly rely on low-level features like shape or color, with other structures like the unsupervised *Restricted Boltzman Machine* (RBM) and the supervised *Convolutional Neural Networks* (CNN), which are instead high-level feature extractors. The results showed that the last two widely overtook the first ones and that their ability to automatically extract discriminative features, without applying any manual feature design, efficiently led to an improvement of the classification. In particular, CNN performed even better than RBM, which demonstrates the advantage of using a supervised feature learning over an unsupervised one. For all the techniques, except for the CNN, a *Support Vector Machine* (SVM) was used as classifier, while for the CNN no separate classifier was needed since it can use directly the final fully-connected layers (see 3.1.2.5). The only problem of using CNNs is that they need a lot of samples to correctly train their parameters, especially when the data do not present - as it often happens with medical images - easy identifiable local and global structures like edges, corners or specific shapes. To overcome this problem, researchers proposed a technique called *Transfer Learning* which exploits the knowledge of networks that have already

been trained using, for example, a large set of labeled natural images. However, the substantial differences between natural and medical images may advise against such knowledge transfer. In [57], *fine-tuning* is presented as method that can help adjusting the pre-trained network parameters in order to make them capable of extracting significative features from medical images, without the need of having a large amount of data available. Specifically, this technique is applied to *AlexNet* network and compared with the same network trained from scratch, both tested on four distinct medical image applications in three specialities (radiology, cardiology and gastroenterology). The research showed that the use of a pre-trained CNN with adequate fine-tuning outperformed or, in the worst case, performed as well as a CNN trained from scratch and that neither shallow tuning nor deep tuning was the optimal choice. Moreover, as it was expected, fine-tuned CNNs proved to be more robust, with respect to training set size, than the network trained from scratch.

The third and last challenge consists in making the extracted features suitable to be classified by, for instance, a SVM. In [19], one of the main brought contributions revisits classical ideas in texture modeling in the light of modern local feature descriptors and *Pooling Encoders*. In particular, they illustrate the benefit of truncating the CNN earlier - still at level of the convolutional layers - in order to obtain powerful local image descriptors that can be combined with traditional pooling encoders, like *Bag of Visual Words* or *Improved Fisher Vector*, to be classified by a SVM. Their results reached the state-of-the-art in recognition accuracy in several benchmarks, including diverse sets of visual domains. The outstanding outcome is that these solutions can be obtained without the need of fine-tuning the CNN parameters, by implicitly reducing the domain shift problem. One of the main findings of this work is that the performance of the structure CNN + FV is often significantly superior than the ones reached with all the other tested structures (e.g. SIFT + FV or CNN + FC) in texture, scene and also object recognition. The difference in the results increases when considering deeper CNN architectures. This can be in part explained by the ability of FV pooling to overfit less and to easily integrate information at multiple image scales. The fact that this structure is substantially less committed to a specific dataset than, for instance, the one using the fully-connected layers, made us think that it would have been the perfect layout to work with in our medical image classification framework.

All these previous works led us to create and test our own proposed solution (4), which aims to put together the techniques mentioned above and create a well-performing classifier for our lung cancer cells images.

Chapter 3

Image Features: Extraction and Pooling

In general, a visual representation is a function that takes as input an image $\mathbf{x}$ and transforms it to a vector $\phi(\mathbf{x})$ in a way that makes its content easier to understand. When facing the problem of supervised classification of images, $\phi(\mathbf{x})$ represents the features of the input image $\mathbf{x}$. As explained in [19], it is useful to divide this map as $\phi = \phi_e \circ \phi_l$, where ϕ_l represents the *Local Descriptors Extractor* and ϕ_e the *Pooling Encoder*.

The first set can be divided in *Hand-Crafted* local descriptors, like *Scale-Invariant Feature Tranform* (SIFT) or *Local Binary Patterns* (LBPs), and *Learned* ones, like *Convolutional Neural Networks* (CNNs). It has already been proven that, concerning image classification problems, the second category overtakes the first one and, because of that, only CNNs are described in detail and used in the thesis.

3.1 Convolutional Neural Networks

The Convolutional Neural Networks are learned-type local descriptor that have had an important role in the history of deep learning. They are a specialized kind of neural network for processing data that has a known grid-like structure, like for example time-series data (1-D grid) and image data (2-D grid).

CNNs are regularized versions of general feedforward neural networks and are able to take advantage of the structural characteristics of the data, like the location of a specific feature, and to use them for assembling more complex patterns. The name "convolutional neural networks" indicates that they exploit a mathematical operation called convolution[1], which is a specialized kind of linear operation used in place of general matrix multiplication in at least one of the network layers.

[1]The name *convolution* is used only by convention: technically speaking, the operation involved is a *sliding dot product* or *cross-correlation* (see [23]). This has significance for the indices in the matrix (it affects how weight is determined at a specific index point), but the idea behind the two operations is exactly the same.

3.1.1 The Convolution Operation

The convolution is an operation on two (real-valued argument) functions that produces a third function expressing how the shape of one is modified by the other. The operator is defined as the integral of the product of the two functions after one is reversed and shifted:

$$(f * g)(t) = \int_D f(\tau)g(t-\tau)d\tau \tag{3.1}$$

where D is the domain of t.
In CNNs terminology, the first argument of the convolution is often referred to as the input, and the second argument as the kernel (in our notation those arguments are respectively f and g). The output is usually referred to as the feature map. Usually, working on computers, the variable t is discretized and in the specific case of 2-D image analysis, the domain D (and so the input of the two functions f and g) is two-dimensional. In this case the equation (3.1) becomes:

$$(f * g)(i,j) = \sum_m \sum_n f(m,n)g(i-m, j-n) \tag{3.2}$$

Discrete convolution can actually be seen as a multiplication by a matrix, which has several entries constrained to be equal to other entries. In two dimensions, this matrix is a specific type of *block circulant matrix* (for more details see [23]), and due to the fact that the kernel is usually much smaller than the input image, as explained in the next section, the matrix turns out to be very sparse.

3.1.2 General Architecture

A CNN is made of an *input* and an *output* layer, and contains various *hidden* layers - determining its depth - which could be of many different types. Let's describe them to understand how they work and how they could be combined one another to form the network structure.

3.1.2.1 Convolutional Layer

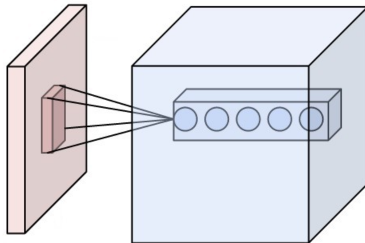


Figure 3.1: General structure of a convolutonal layer (taken from [6] and slightly modified). The depth of the output depend on the number of channels we set for the layer (hyperparameter).

This type of layer is the core building block of a convolutional neural network. The input is a $n \times m \times d$ image, where n and m are respectively the width and the height of the image, and d its *depth* (e.g. when considering an *RGB* input image, d is equal to 3). The convolutional layer's parameters consist of a set of learnable filters (or kernels), which have a spatially small receptive field in width and height, but extend on the full depth of the input volume. During the forward pass, we convolve each filter across width and height of the input image, which in practice is equal to compute the dot product between the input and the entries of the filter (see Figure 3.1). This will produce a 2-dimensional *activation map* and, as result, the network learns how to detect specific *features* at some spatial position. A complete convolutional layer is composed of several feature maps (with different weight vector[2]), so that multiple features can be extracted at same location. To ensure some degree of shift, scale and distortion invariance, the CNNs combine three structural ideas (as explained in [32]):

- **Local Receptive Fields**: it is the region of the input image at which the filter is connected. It depends on the size of the filter, which is a hyperparameter setted *a priori*, and on the weights assigned to each connection, which are learned during the training of the model. Since each neuron is connected to a small region of the input image, the network is able to take into account its spatial structure and the localization of its features.

- **Shared Weights**: as already said, the filters act on a certain receptive field and "move" through the input image. If during this process the weights of the filter in a particular layer do not change, we say that we are sharing weights. The idea is that if a specific feature (like an edge) is important to be learned in a particular part of the image, it may probably be important also in other parts. The great advantage of using shared weights is that it is possible to substantially lower the degrees of freedom of the problem (i.e. the number of parameters to learn and optimize), which also implies a faster convergence to some minimum. Moreover, this property has the effect of making the model less flexible, but this actually works as a regularizer and helps avoiding *overfitting*, since the weights are shared among neurons.

- **Spatial Arrangement**: three hyperparameters control the size of the convolutional layer's output:

  - *Depth*: it corresponds to the number of filters we would like to use and controls the number of neurons in a layer that connect to the same region of the input volume. If for example the input is the raw image, different neurons along the depth dimension may activate in the presence of various blobs of color or oriented edges.

[2] The function that is applied to the input values is determined by a vector of weights and a bias (typically real numbers), called *filter*. The learning process of the network consists in adjust iteratively these filters.

– *Stride*: it controls how to allocate the result of the convolution around the width and height. When the stride is 1 then we move the filters one pixel at a time. Depending on the *stride*, we could find heavily overlapping receptive fields between the columns, and also large output volumes.

– *Zero-Padding*: sometimes it results useful to *pad* with zeros the border of the input image. Padding allows us to control the output volume spatial size, which is often preserved in the convolutional layers.

The spatial size of the output volume can be computed as follows:

$$\frac{W - F + 2P}{S} + 1 \tag{3.3}$$

where W is the input volume size, F is the filter (kernel field) size of the convolutional layer neurons, S is the the stride with which they are applied and P is the amount of zero-padding used at the border. If the number given from the formula (3.3) is not an integer, it means the strides are not correct to make the neurons fit the input in a symmetric way.

3.1.2.2 Pooling Layer

Another important component of CNNs are the pooling layers, which are used to perform a non-linear down-sampling (see Figure 3.2). As explained in [53], their purpose is to achieve spatial invariance by reducing the resolution of the feature maps.

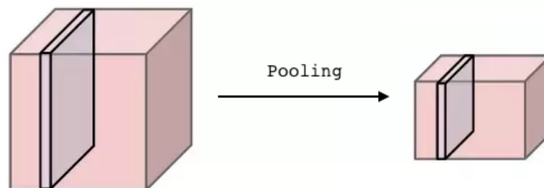


Figure 3.2: General structure of a pooling layer (taken from [6] and slightly modified).

Their units combine the input from a small $n \times n$ patch of units. This pooling window can be of arbitrary size and different windows can be overlapping (depending on the stride). Pooling layers operate independently on every depth slice of the input and serve to progressively reduce the spatial size of the representation, i.e. the number of parameters. They also help controlling memory waste and finally the overfitting on training data. It is common to periodically insert a pooling layer between successive convolutional layers in a CNN architecture. Intuitively, the idea behind the use of pooling is that the exact location of a feature is less important than its rough location relative to other features.

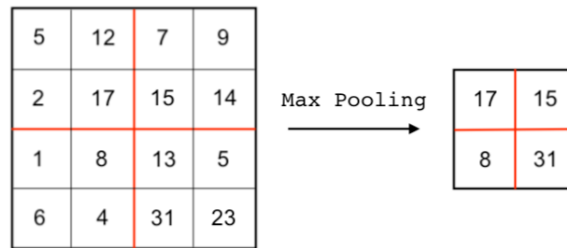


Figure 3.3: 2×2 max pooling window with stride 2, on a 4×4 input image. It partitions the input image into a set of non-overlapping squares[3]and, for each sub-region, outputs the maximum.

There are several non-linear functions to implement pooling, among which *max pooling* (shown in Figure 3.3) and *average pooling* are the most commonly used.

3.1.2.3 Activation Layer

As known from theory, the output y of a *Linear Neuron* is represented as:

$$y = \sum_i x_i w_i + b \tag{3.4}$$

where i is the index over the input connections, x_i and w_i are respectively the ith elements of input vector $\mathbf{x}$ and weight vector $\mathbf{w}$, and b is the bias. The activation function $\psi : \mathbb{R} \to \mathbb{Y}$ acts as follows:

$$y = \psi \left(\sum_i x_i w_i + b \right) \tag{3.5}$$

and its role is to decide whether a neuron should be activated or not. Its purpose is to introduce non-linearity into the output of a neuron and to help normalize it (usually $\mathbb{Y}$ is set equal to $[0, 1]$ or $[-1, 1]$). The simplest activation function is the *Heaviside* (see Figure 3.4):

$$\psi(x) = \begin{cases} 0 & x < t \\ 1 & x \geq t \end{cases} \tag{3.6}$$

This function imposes a threshold t that determines if the neuron is activated or not, but presents some problems in classification with more than two classes. One of the most used activation function is the so called *Sigmoid* function (Figure 3.5):

$$\psi(x) = \frac{1}{1 + e^{-x}} \tag{3.7}$$

[3]The shape of the window could also be a rectangle: "Region of Interest" (RoI) pooling is a max pooling variant, where output size is fixed and input rectangle is a parameter (details in [24]).

This function presents several advantages: it is non-linear, its output is in the range $[0, 1]$ (i.e. the activation cannot "blow up"), it can do not-binary classification and it has a very smooth gradient[4].
The biggest problem that occurs with sigmoid activation function is that it gives rise to a phenomenon called "vanishing gradient". The sigmoid has the tendency to bring the values of y to either the end or the beginning of the curve (i.e. any small change in the values of x, in the region around 0, will cause values of y to change significantly), which means the gradient in those regions is very small and y tends to respond very less to changes in x. The main consequence is that the network "refuses" to learn further or drastically slows down.

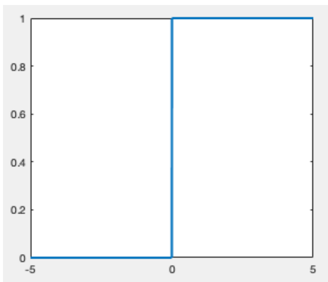


Figure 3.4: Heaviside

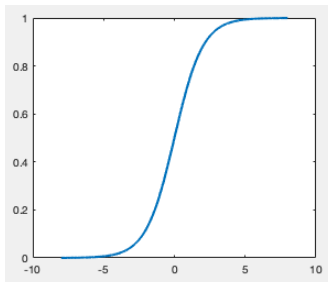


Figure 3.5: Sigmoid

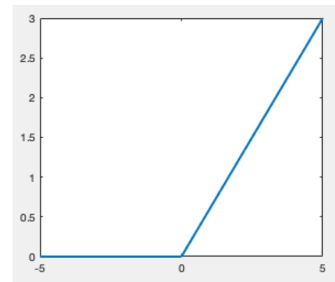


Figure 3.6: ReLU

The most common activation function used in convolutional neural networks is called *ReLU* function (*Rectified Linear Unit*), it is shown in Figure 3.6 and it is mathematically equal to:

$$\psi(x) = \max(0, x) \tag{3.8}$$

The biggest advantage of the ReLU activation is its efficiency: because of its structure, it does not let all the neurons "fire" (sparse activation) and the network results lighter (unlike the sigmoid function, with which the activation is dense and it is very costly). Unfortunately, the ReLU activation function also presents some disadvantages. The first one is that its output range is $[0, +\infty]$ and, since it is not bounded, it could "blow up" the activation, which computationally represents a limit. Its second weakness is the so-called *dying ReLU problem*: because of the horizontal line for negative x, the gradient go towards 0 and stops adjusting the weights during descent. This means that the neurons going into that state will stop responding to variations in error/input. This problem could cause many neurons to just "die", making a not negligible part of the network passive. There are variations of the ReLU to mitigate this issue by simply imposing a little slope on the horizontal part. This technique is called *Leaky ReLU* (see [65]).

[4] This result is very important during the *backpropagation* step.

3.1.2.4 Dropout Layer

The *dropout* is a regularization technique used in neural networks learning problems (e.g. with a CNN architecture) that generally leads to a reduction of the network overfit on the training set. This is done by approximating the train over a large number of neural networks with different architectures in parallel ([3]).
In practice, the dropout layer could be used with most types of layers, such as convolutional or dense fully connected layers, but not the output one.

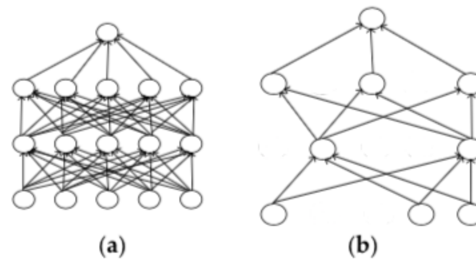


Figure 3.7: Here are shown the connections in a simple neural network structure.
(a) Without using the dropout layer. (b) Using the dropout layer.

During the training phase this method randomly *drops-out*, according to certain probabilities[5], some nodes of the network (see Figure 3.7), in order to ignore them and to avoid the fact that they generally appear multiple times simultaneously in the considered model. In this way, each update to a layer during training is performed with a slightly different network and, consequently, with a different point of view on the configured layer. The final result is obtained by performing the weighted average of all the different models.
This technique is computationally cheap and very effective as a regularization method. Its consequences, in addiction to overfitting reduction, are:

- It creates a noise in the training process, making the nodes less or more responsible for the inputs.
- It improves the generalization error, i.e. the measure of how well a classification algorithm is able to classify unseen data.
- It leads to a more sparse training representation.

Concluding, the dropout layer usually tends to improve the network performance when applied, but slightly increases the processing time in the learning stage.

[5]This is a hyperparameter which coincides with the probability to retain or not a node.

3.1.2.5 Fully Connected and Loss Layers

The fully connected layer is an essential component of CNNs and have been proven very useful in computer vision[6]. Its role is to classify the original input image into a label using the output of the conv/pool layers. This output is flattened, i.e. converted into a 1-dimensional vector of values, before entering the FC layer (see Figure 3.8). A standard CNN structure can have more than one FC layer, but the important thing is that the output of the last one has dimension equal to the number of labels. The output of the last FC layer, for each image $x_i \in X = \{x_1, ..., x_I\}$, is a vector $\boldsymbol{y}_i \in \mathbb{R}^C$ where C is the number of labels and every real value $(\boldsymbol{y}_i)_k$ in the vector is the kth class score for the ith image.

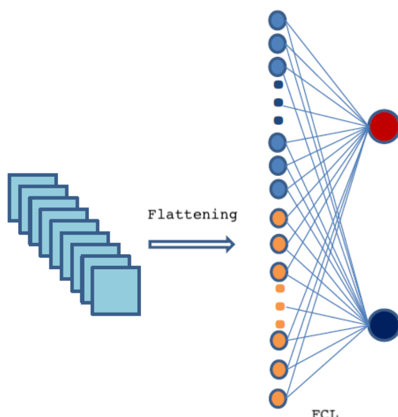


Figure 3.8: Flattening and fully connected layer with two final classes.

After this, a loss layer is applied, which is always the last layer of the neural network. Its role is to adjust the weights (initially setted at random) across all the network, going through the backpropagation process (1.2.1.3). There exist many types of loss functions and each one is appropriate for different tasks. Its choice can be a challenging issue as the function must capture the properties of the problem and be motivated by concerns that are important to the project. One of the most used with CNNs is the *Log-Loss* (also called *Cross Entropy* in information theory), which can predict a single class in a set of mutually exclusive classes. More precisely, it is based on the concept of *Maximum Likelihood Estimation* (MLE), which is a method of estimating the parameters of a probability distribution by maximizing a likelihood function over the parameters space. Let $X = \{x_1, ..., x_I\}$ be the images dataset and $Y = \{\boldsymbol{y}_1, ..., \boldsymbol{y}_I\}$ the output of the model, which follows a distribution with density function f depending on a parameter w. The likelihood function is then defined as follows, both for a single

[6]The fully connected layer can be seen as an order-sensitive pooling encoder (3.2), but because of its low flexibility it is exploitable just with CNNs. That's why it is described in this section.

sample and for all the training set:

$$p_w(\boldsymbol{y}_i; x_i) = f(\boldsymbol{y}_i|w) \quad \rightarrow \quad p_w(Y;X) = \prod_{i=1}^{I} f(\boldsymbol{y}_i|w) \tag{3.9}$$

In other words, the likelihood is equal to the product of probability densities of every outcome $\boldsymbol{y}_i$ when the true value of the parameter is w. The problem of maximizing this function should intuitively select the parameter values that make the observed data most probable. In practice, it results helpful to work with the natural logarithm of the likelihood (log-likelihood) since it presents some nice properties as monotonicity and it generally simplifies the representation of very big numbers. In classification tasks, it is common to treat MLE as a minimization problem, since the loss function should represent the cost committed in assigning wrong labels to samples. That's why the used function is the so-called *negative log-likelihood* (NLL). In formula, after some computations:

$$L(w) = -\sum_{i=1}^{I} \ln\left(\, p_w(\boldsymbol{y}_i)\,\right) \tag{3.10}$$

where $\{\boldsymbol{y}_i\}_{i=1}^{I}$ is the output of the model, i.e. of the last FC layer. This function coincides exactly with the *Log-Loss* applied in the loss layer of the CNN.
Since each output $\boldsymbol{y}_i$ of the FC layer is a vector of real numbers, we need to apply a transformation to interpret them as belonging probabilities to each class. We introduce then the *Softmax* function $\sigma : \mathbb{R}^C \rightarrow \mathbb{R}^C$, defined as:

$$\sigma(\,(\boldsymbol{y}_i)_k\,) = \frac{e^{(\boldsymbol{y}_i)_k}}{\sum_{j=1}^{C} e^{(\boldsymbol{y}_i)_j}} \qquad k = 1, ..., C \tag{3.11}$$

After being normalized with the softmax function, all the components of $\boldsymbol{y}_i$ will belong to the interval $(0, 1)$ and will add up to 1. That's why they can be interpreted as probabilities. The result of $\sigma(\boldsymbol{y}_i)$ is what in the expression of the loss function is indicated as $p_w(\boldsymbol{y}_i)$.

3.1.3 Learning Techniques

In this section are presented two different learning approaches that exploit all the concepts presented so far. The first one is the *Training from Scratch* learning, in which the CNN is trained for the first time on the target dataset. This method generally needs a lot of data to reach high accuracy performances, but it has the advantage of leaving to the user the choice of the most appropriate network structure, depending for instance on the amount of samples and on the images characteristics.
The second approach is the so-called *Transfer* learning. The basic idea here is that in literature there exist network structures that have already been successfully trained on huge amount of data and that, with slight modifications, could be reused to classify

images taken from a new dataset. The biggest asset of this approach is that it usually does not need a big amount of samples to perform well.
Both these approaches present weaknesses that, in real-world problems, could lead to bad classification results. Because of that, we also present here some methods for improving their classification performance.

3.1.3.1 Train from Scratch

Training a network from scratch needs the user to take a lot of decisions that could lead the performance to sensibly grow or decrease. The first one, and probably also the most important, is the choice of the network structure. In section 3.1.2 we described the different layer types of a CNN and their role inside the network, but not how to efficiently link them together. Unluckily, in literature there are no theoretical rules that leads to an always well-performing network, but only some guidelines. The most part of the effort in selecting the structure that best fits a certain problem is done empirically. In the last decades, many different architectures were tested on many different problems to expand the empirical knowledge baggage and to reach good classification performances even without solid theoretical foundations.
After deciding the suitable network structure, we need to set the regularization techniques that help avoiding overfitting and also some important hyperparameters, like the number of epochs, the learning rate and the gradient descent batch size. The three main regularization techniques commonly applied are the already described dropout layer (3.1.2.4) and the L^1 (*Lasso*) and L^2 (*Ridge*) penalty regularization of the loss function. Another technique that can be thought of as a regularization one is *data augmentation*, which usually results in an increasing of the accuracy when the training set does not contain many observations.
The learning of the CNN consists in the updating of the layers weights through the backpropagation process in order to minimize the value of the loss function and, in some way, to make the network "understand" what are the important patterns to be recognised in the input images.

Data Augmentation

It is of common knowledge that the more data a learning algorithm has access to, the greater can be its effectiveness. Data augmentation is a technique that allows to increase the amount of images in the training set[7], without actually collecting new data, by applying transformations that are class-invariant for the original images. Some example of transformations are: *rotation, rescale, flip, crop, shift* and *gaussian noise.*
The class-invariance of these transformations strongly depends on the type of data that is being analyzed, reminding that the goal is to not increase the *irrelevant data*, i.e. to

[7] It is actually possible to use data augmentation also for the test sample to make the prediction more robust.

use transformations that do emphasize the important features of the data structure. As described in [40], data augmentation give us another not negligible advantage: it represents a way we can reduce model *overfitting*. This is done increasing the amount of training data using information contained only in our training data. The model performance increases by reducing data bias and improving model generalization. When the original dataset is small the augmented images can be stored directly in memory; this technique is called *offline* data augmentation. If the dataset is large we can instead apply the so-called *online* data augmentation, which consists in considering augmented batches of data and fed the network with them.

Cost-Sensitive Learning

The *cost-sensitive* learning is based on the idea that the misclassification errors of a model have to be weighted differently depending on the membership to a certain class. As the *Precision-Recall curve* (1.2.3.1), this technique helps to deal with the problem of imbalanced classes, but with the difference that it is applied just during the CNN training phase (both when training from scratch and when fine-tuning the parameters in transfer learning), while the PR curve it is used after the classification with the SVM, regardless of how the local descriptors are extracted. In real-world binary classification problems exist many cases in which the actual "cost" of misclassifying a sample from the positive class is bigger than misclassifying the negative ones (e.g. fraud/cancer detection), which in practice lead to the definition of the so-called *cost matrix* C:

$$C = \begin{bmatrix} c_{(++)} & c_{(+-)} \\ c_{(-+)} & c_{(--)} \end{bmatrix} \tag{3.12}$$

where the rows represent the predicted classes and the columns the actual ones (to see the related confusion matrix check Figure 1.10). The elements of this matrix, if possible, are set through prior knowledge - i.e. an expert opinion on how it would cost to misclassify each class - but sometimes happens that this information is not available. In this case the cost is taken inversely proportional to the number of samples in each class. To better understand this, suppose the positive class contains 100 samples while the negative one just 1. The cost $c_{(+-)}$ of misclassifying a true negative is set to 100, while $c_{(-+)}$ to 1. The costs of correctly classifying positive and negative samples, respectively $c_{(++)}$ and $c_{(--)}$, are usually set to 0. All the assigned costs are used in the total cost expression to give more or less importance to certain misclassified samples with respect to others. This is done weighting the misclassified samples in the loss function with respect to their class of belonging ([47]). To heuristically apply this technique during the CNN training, we first create a vector of weights that is then passed to the loss layer (3.1.2.5), which will multiply every sample by its corresponding class weight before computing the loss function. This is a particular way to apply cost-sensitivity, called *weighted learning*.

3.1.3.2 Transfer Learning

This technique is one of the most common and most used in *machine learning*. Described in a nutshell, the focus of this method is on the reuse of some type of knowledge, obtained solving a certain problem, to a new and different issue, which is somehow related to the first one. More precisely, it consists in taking the parameters of an already trained network (generally on a very large dataset, like *ImageNet*) to exploit its feature extraction topology in a new classification problem.
As explained in [58], this need arises from the fact that the collection of new samples could be very difficult and expensive in real world applications and that an insufficient number of training data is an inescapable issue in some special domains. It seems then very useful to use, as a starting step, good classification results that have already been reached, with the goal of successfully learn a new task with a very good performance and in less time. Generally speaking, the training data must be *independent and identically distributed* (i.i.d.) with the test data, but using *Transfer Learning* it is possible to relax this hypothesis, which means that the model in target domain does not need to be trained from scratch. This helps to reduce significantly the demand of training data and training time in the new domain. Another advantage of using this technique is that it also deals with the problem of *overfitting*. This is because the transferred network is trained on a different set of data and this, in a certain way, makes it less "expressive" with respect to the new one. This means that the model does not produce an analysis that corresponds too closely or exactly to the new set of data to which it is applied. Usually, the architecture of pre-trained networks used for image classification is specific for a certain type of dataset (e.g. dimension and type of the input images or the number of labels in which the network is classifying) and so, in order to make them work, some modification to their architecture or to the dataset are needed. In general, the features are extracted from the layer that acts before the classification ones, but this is not always true and depends also on the complexity of the network with respect to the new dataset (more precisely, if the network is very deep, while the used images are small and "simple", the features could be extracted from an earlier layer).

Parameters fine-tuning

With transfer learning, the best results are usually found when the dataset on which the network was originally trained and the dataset of our problem are very similar. If this does not happen, or if we just want to try to increase the performance of our model, it is possible to fine-tune the parameters of the pre-trained network and "adapt" them to the new dataset. This is done re-training the network on the images of interest, using again the backpropagation algorithm and the batch gradient descent to minimize the loss function and to update the network weights.
Given the L layers from the pre-trained network, this technique could be applied in different ways:

- *Entirely*: it is possible to train again all the L pre-trained layers, using their weights as initialization for the new training process.

- *Partially*: in this case we could just choose to re-train the first or last $N < L$ layers, in order to model the new distribution of data for the new task, or we could add new layers on top of the N pre-trained ones and train them from scratch. In both cases, we need to "freeze" the layers that we don't want to re-train and make the other ones learn faster. This is done by changing the value of their *Learning Rate* in order to accelerate the parameters adjusting.

The fine-tuning process does not always lead to an improvement in the classification performance of the model, but compared with a network trained from scratch, it is generally more robust to the size of the training set and, if used adequately, it usually performs better ([57]).

3.2 Pooling Encoders

The idea of this work is not to directly make use of the CNN as a classifier, but to exploit its characteristics in order to efficiently extract features from the lung cancer cell images[8]. These features are collected and organized in the form of local descriptors. The role of *pooling encoders* is to take as input the extracted local descriptors and to produce a single feature vector which aims to be well suitable for tasks as classification with, for example, a linear *Support Vector Machine* (SVM, see 1.2.2).
A smart way to divide encoders is considering whether they take into account the spatial configuration of input features or whether they discard it (i.e. if the map ϕ_e, introduced at the beginning of this chapter, is invariant or not to some permutation of the input). This two different types of pooling encoders are respectively called *Order-Sensitive* and *Orderless* ones. An example from the first family of pooling encoders is the already described *Fully-Connected Layer*, while from the second one we are going to describe the *Fisher Vector*, the *Bag of Visual Words* and the *Vector of Locally-Aggregated Descriptors*. In our specific experiment framework, we present their classification performances in chapter 5.

3.2.1 Fisher Vector

The *Fisher Vector* (FV) is an image representation obtained by pooling local image features. Its encoding process is based on the concept of *Fisher Kernels* and its adaptation to image classification ([44], [52]).

[8] When the dataset is not so big, as in our case, it would be better to extract features from an earlier layer of the convolutional stage, as this will be set in more general patterns than the later ones.

3.2.1.1 The Fisher Kernel

The *Fisher Kernel* (FK) is a map that gives a 'natural' similarity measure which takes into account an underlying probability distribution. As the name suggests, this function derives a kernel which is directly extracted from a generative model of the data. In the case of FV, this model coincides with a *Gaussian Mixture Model* (GMM) and can be interpreted as a "universal probabilistic visual vocabulary". In a few words, the FK characterizes the samples by their deviation from the GMM[9].
Let $X = \{x_i, i = 1, ..., I\}$ be a sample of I observations $x_i \in \mathbb{R}^D$. Let p_λ be the probability density function which models the generative process of the observations. $\lambda = [\lambda_1, ..., \lambda_M]' \in \mathbb{R}^M$ is the vector of the M parameters of p_λ. Note that in our case p_λ coincides with a weighted linear combination of Gaussians.
In statistics, the *Fisher score function* is defined as:

$$G_\lambda^X = \nabla_\lambda \log p_\lambda(X). \tag{3.13}$$

It indicates the *slope* of the log-likelihood function evaluated in a specific point of the parameter vector λ. Put differently, it represents how the parameters vector λ should be modified to better fit the data X. Note that $G_\lambda^X \in \mathbb{R}^M$, and hence its dimensionality does not depend on the size I of the samples set.
Let's define the *Fisher Information Matrix* (FIM) as in [11]:

$$F_\lambda = \mathbb{E}\big[G_\lambda^X \, G_\lambda^{X'} \big| \lambda\big] = \mathbb{E}_{x \sim p_\lambda}\big[G_\lambda^X \, G_\lambda^{X'}\big] \tag{3.14}$$

with $F_\lambda \in \mathbb{R}^{M \times M}$. This matrix measures the information about the parameters λ that is contained in the samples in X. As proposed in [27], the *Fisher Kernel* can be seen as the similarity measure between two samples X_i and X_j as follows:

$$K_{FK}(X_i, X_j) = G_\lambda^{X_i'} \, F_\lambda^{-1} \, G_\lambda^{X_j} \tag{3.15}$$

Let's now note that, since F_λ is positive semi-definite, its inverse exists and it is positive semi-definite too. Hence, we can make use of the following *Cholesky* decomposition:

$$F_\lambda^{-1} = L_\lambda' \, L_\lambda \tag{3.16}$$

and finally rewrite the kernel as the new dot-product:

$$K_{FK}(X_i, X_j) = G_\lambda^{X_i'} \, L_\lambda' \, L_\lambda \, G_\lambda^{X_j} = \mathcal{G}_\lambda^{X_i'} \, \mathcal{G}_\lambda^{X_j} \tag{3.17}$$

The normalized gradient vector $\mathcal{G}_\lambda^X \in \mathbb{R}^M$ is called *Fisher Vector* of X. The biggest advantage of this formulation is that linear classifiers can efficiently learn from this

[9] In a general classification procedure, the class for a test object can be estimated by minimising, across classes, an average of the FK distance (which is equivalent to maximize the average of the FK similarity) from each known member of the given class to the new object.

representation, since a non-linear kernel machine using K_{FK} as kernel is equivalent to a linear kernel machine using the FV as feature vector. This is beneficial because training a linear classifier is way less computationally expensive than training a non-linear one.

3.2.1.2 Application to Images

Let now $X = \{x_i,\ i = 1, ..., I\}$, with every $x_i \in \mathbb{R}^D$, be the local descriptors extracted from an image (e.g. with CNN). Substituting (3.17) in (3.13) and assuming independence between samples[10], it is possible to rewrite the FV as:

$$\mathcal{G}_\lambda^X = \sum_{i=1}^{I} L_\lambda \nabla_\lambda \log p_\lambda(x_i). \tag{3.18}$$

Under this assumption, the FV coincides with a sum of normalized gradient statistics computed for each local descriptor. This statistics embed the extracted features in a higher-dimensional space where, as showed in Figure 1.9, they are more suitable to be separated by a linear classifier. As already mentioned, we choose the p_λ to be a GMM and we denote its parameters by $\lambda = \{w_k, \mu_k, \Sigma_k\}$ with $k = 1, ..., K$, where w_k represents the mixture weight of Gaussian k, μ_k its mean vector and Σ_k its covariance matrix. We can now write:

$$p_\lambda(x) = \sum_{k=1}^{K} w_k f_k(x) \tag{3.19}$$

where f_k denotes the density function of the D-dimensional Gaussian k:

$$f_k(x) = \frac{1}{(2\pi)^{D/2}|\Sigma_k|^{1/2}}\, exp\left(-\frac{1}{2}\,(x-\mu_k)'\,\Sigma_k^{-1}\,(x-\mu_k)\right). \tag{3.20}$$

To actually make sure $p_\lambda(x)$ is a distribution, the following constraints are imposed on the weights:

$$\forall k:\ w_k \geq 0 \quad , \quad \sum_{k=1}^{K} w_k = 1 \tag{3.21}$$

In this representation, the matrix Σ_k is supposed diagonal $\forall k$[11], and we call σ_k^2 the variance vector, corresponding to the diagonal of Σ_k. The parameters of the GMM are estimated on the training set of local descriptors (usually a subset Y of X) using the *Expectation-Maximization* (EM) algorithm to optimize the maximum likelihood criterion. To obtain the expressions of all the normalized gradients we first need to

[10] Concerning images, this assumption is generally incorrect, but it is possible to deal with it as explained in the next section on FV normalization.

[11] This reduces significantly the number of parameters to learn and is usually an acceptable compromise in vision applications. If the data is significantly correlated, it can be beneficial to de-correlate it by PCA rotation or projection in pre-processing.

adopt the following re-parametrization:

$$w_k = \frac{\exp(\alpha_k)}{\sum_{j=1}^{K} \exp(\alpha_j)} \tag{3.22}$$

and so the new parameter vector is $\lambda = \{\alpha_k, \mu_k, \Sigma_k\}$. Once set this, we introduce a quantity called *soft assignment* of x_i to the Gaussian k (or posterior probability):

$$\gamma_i(k) = \frac{w_k f_k(x_i)}{\sum_{j=1}^{K} w_j f_j(x_i)} \tag{3.23}$$

It is important to specify that in all this section the operations of division, exponentiation, power or square root of vectors must be considered as term-by-term.
The last remaining step is computing the *Cholesky* decomposition matrix: $L_\lambda = F_\lambda^{-\frac{1}{2}}$. As shown in Appendix **A** of [52], under some assumption on γ_i it is possible to consider the matrix F diagonal. Taking this into account and after some math calculation we find the three normalized gradients:

$$\mathcal{G}_{\alpha_k}^X = \frac{1}{\sqrt{w_k}} \sum_{i=1}^{I} (\gamma_i(k) - w_k) \tag{3.24}$$

$$\mathcal{G}_{\mu_k}^X = \frac{1}{\sqrt{w_k}} \sum_{i=1}^{I} \gamma_i(k) \left(\frac{x_i - \mu_k}{\sigma_k^2} \right) \tag{3.25}$$

$$\mathcal{G}_{\sigma_k^2}^X = \frac{1}{\sqrt{2 w_k}} \sum_{i=1}^{I} \gamma_i(k) \left[\frac{(x_i - \mu_k)^2}{\sigma_k^2} - 1 \right] \tag{3.26}$$

The resulting fisher vector $\mathcal{G}_\lambda^X$ is just the concatenation of these three normalized gradients. Note that to avoid the dependence on the sample size, the final FV is normalized by the coefficient I (in the algorithm 1 this sample size normalization is already performed in the gradients calculation step): $\mathcal{G}_\lambda^X \leftarrow \frac{1}{I} \mathcal{G}_\lambda^X$.
The algorithm presented could also be expressed, and consequently solved, in terms of the following statistics, with $S_k^0 \in \mathbb{R}$, $S_k^1 \in \mathbb{R}^D$ and $S_k^2 \in \mathbb{R}^D$:

$$S_k^0 = \sum_{i=1}^{I} \gamma_i(k) \quad , \quad S_k^1 = \sum_{i=1}^{I} \gamma_i(k) x_i \quad , \quad S_k^2 = \sum_{i=1}^{I} \gamma_i(k) x_i^2 \, . \tag{3.27}$$

3.2.1.3 FV Normalization

The normalization of the FV is a necessary step to obtain good results using a linear classifier. The methods used in this thesis and proposed by [45] act as follows.
$\mathcal{L}^2$-normalization: in image processing can happen that different images contain different amounts of background information. Let's assume that given an image, its descriptors X follow a distribution d. Thanks to the *law of large numbers* and to eq.

(3.13) we obtain:

$$\frac{1}{I}G_\lambda^X \approx \nabla_\lambda \mathbb{E}_{x\sim d} \log p_\lambda(x) = \nabla_\lambda \int_x d(x) \log p_\lambda(x) dx \tag{3.28}$$

Suppose it is possible to decompose d in an image-specific part which follows a certain distribution q, and an image-independent (background) part which follows p_λ:

$$d(x) = \omega q(x) + (1-\omega) p_\lambda(x) \tag{3.29}$$

with $0 \leq \omega \leq 1$ representing the proportion of image-specific information inside the image. If we try to estimate λ maximizing $\mathbb{E}_{x\sim d} \log p_\lambda(x)$, at least locally and approximately, we can write:

$$\nabla_\lambda \int_x p_\lambda(x) \log p_\lambda(x) dx = \nabla_\lambda \mathbb{E}_{x\sim p_\lambda} \log p_\lambda(x) \approx 0 \tag{3.30}$$

Substituting first eq. (3.29) and then eq. (3.30) in (3.28) we obtain:

$$\frac{1}{I}G_\lambda^X \approx \omega \nabla_\lambda \int_x q(x) \log p_\lambda(x) dx \tag{3.31}$$

This shows that the background information can be approximately discarded by the FV. To finally get rid of the dependence on ω we can just apply the $\mathcal{L}^2$-normalization to the FV[12]. The operation of normalizing the FV could also be interpreted in terms of changing the *kernel* in (3.17) as:

$$K(X_i, X_j) = \frac{K(X_i, X_j)}{\sqrt{K(X_i, X_i) K(X_j, X_j)}} \tag{3.32}$$

Power Normalization: empirically speaking, it has been noticed that the more the number of Gaussians increases, the fewer descriptors x_i are assigned with a significant probability $\gamma_i(k)$ to each Gaussian. This means, in other words, that the fisher vectors tends to become sparse. Remind that, on $\mathcal{L}^2$-normalized vectors, the dot product coincides with the L^2 distance and that this is not a good measure of similarity on sparse vectors. Let's then introduce the following transformation:

$$g(z) = sign(z)|z|^p \tag{3.33}$$

with $0 \leq p \leq 1$. This function is applied to each dimension of the fisher vectors, right before the $\mathcal{L}^2$-normalization step, in order to reduce the sparsity of the representation. The value p represents a hyperparameter and is chosen to be equal to 0.5. Because of this, in this specific case the transformation in (3.33) is also called *Signed Square Rooting*.

[12] Actually this would work normalizing by any $\mathcal{L}^p$-norm. The value $p = 2$ is justified from the fact that it is the natural norm associated with the dot product.

The algorithm to compute the normalized FV - also called *Improved Fisher Vector* (IFV) - from the local descriptors is resumed in the algorithm 1.

Algorithm 1: Improved Fisher Vector (IFV) Algorithm

Input:
- GMM parameters: $\lambda = \{w_k, \mu_k, \Sigma_k; k = 1, ..., K\}$
- Local Image Descriptors: $X = \{x_i\}_{i=1}^{I}, \; x_i \in \mathbb{R}^D \; \forall i$

Output:
- Normalized Fisher Vector: $\mathcal{G}_\lambda^X \in \mathbb{R}^{K(2D+1)}$

1. **Expectation Maximization algorithm for the GMM**
   - Introduce α_k and $\gamma_i(k)$, $\forall k = 1, ..., K$:

$$w_k = \frac{\exp{(\alpha_k)}}{\sum_{j=1}^{K} \exp{(\alpha_j)}} \quad , \quad \gamma_i(k) = \frac{w_k f_k(x_i)}{\sum_{j=1}^{K} w_j f_j(x_i)}$$

   - Compute the normalized gradients:

$$\mathcal{G}_{\alpha_k}^X = \frac{1}{I\sqrt{w_k}} \sum_{i=1}^{I} (\gamma_i(k) - w_k)$$

$$\mathcal{G}_{\mu_k}^X = \frac{1}{I\sqrt{w_k}} \sum_{i=1}^{I} \gamma_i(k) \left(\frac{x_i - \mu_k}{\sigma_k^2} \right)$$

$$\mathcal{G}_{\sigma_k^2}^X = \frac{1}{I\sqrt{2w_k}} \sum_{i=1}^{I} \gamma_i(k) \left[\frac{(x_i - \mu_k)^2}{\sigma_k^2} - 1 \right]$$

   - Concatenate all the FV components:

$$\mathcal{G}_\lambda^X = (\mathcal{G}_{\alpha_1}^X, ..., \mathcal{G}_{\alpha_K}^X, \mathcal{G}_{\mu_1}^{X\,\prime}, ..., \mathcal{G}_{\mu_K}^{X\,\prime}, \mathcal{G}_{\sigma_1^2}^{X\,\prime}, ..., \mathcal{G}_{\sigma_K^2}^{X\,\prime})$$

2. **Normalization of the FV**
   - *Signed Square Rooting* normalization $\forall i = 1, ..., K(2D+1)$:

$$[\mathcal{G}_\lambda^X]_i \leftarrow sign([\mathcal{G}_\lambda^X]_i) \sqrt{|\,[\mathcal{G}_\lambda^X]_i\,|}$$

   - $\mathcal{L}^2$-normalization:

$$\mathcal{G}_\lambda^X = \mathcal{G}_\lambda^X \,/\, ||\mathcal{G}_\lambda^X||_{\mathcal{L}^2}$$

3.2.2 Bag of Visual Words

In computer vision, the *Bag of Visual Words* (BoVW) is an encoding technique, first introduced in [34] and then widely used in many classification problems, where the (frequency of) occurrence in a vocabulary, of each detected feature, is used for training a classifier (in this thesis a linear SVM classifier).
To explain how the vocabulary is constructed and how the BoVW works, we first need to introduce the famous clustering algorithm called *K-means*.

3.2.2.1 The K-means Algorithm

The *K-means* algorithm was theoretically introduced in 1956 by H. Steinhaus ([56]) and is still one of the most used and well-performing in *unsupervised* machine learning to solve clustering tasks. In short words, the algorithm objective is to partition the observations and assign each of them to one of K clusters, using a specific association technique.
Given the set of observations $X = \{x_i,\, i = 1, ..., I\}$, with $x_i \in \mathbb{R}^D\ \forall i$, the algorithm tries to divide them in K sets, with K obviously smaller than I, called clusters and indicated as:

$$\mathcal{S} = \{S_1, ..., S_K\} \tag{3.34}$$

Its objective is to minimize, over all the possible sets of clusters $\mathcal{S}$, a quantity called *Within-Cluster Sum of Squares* (WCSS) that represents the variance of the clusters and is defined as:

$$WCSS\,(\mathcal{S}) = \sum_{k=1}^{K} \sum_{x_j \in S_k} ||\, x_i - c_k \,||^2 \tag{3.35}$$

To do that, the algorithm is divided in two essentials steps. In the first one, we initialize a set of cluster *centers* or *means* c_k, with $k = 1, ..., K$, and all the data points are *assigned* to the cluster with the nearest center. To evaluate the distance between the centers and the samples many distances could be used. For completeness, here are reported just the two most used possibilities[13]:

$$\begin{aligned} &\text{Euclidian Distance:} \qquad d(x_i, x_j) = \sqrt{\sum_{d=1}^{D} \big(\, [x_i]_d - [x_j]_d \,\big)^2} \\ &\text{Manhattan Distance:} \qquad d(x_i, x_j) = \sum_{d=1}^{D} \Big|\, [x_i]_d - [x_j]_d \,\Big| \end{aligned} \tag{3.36}$$

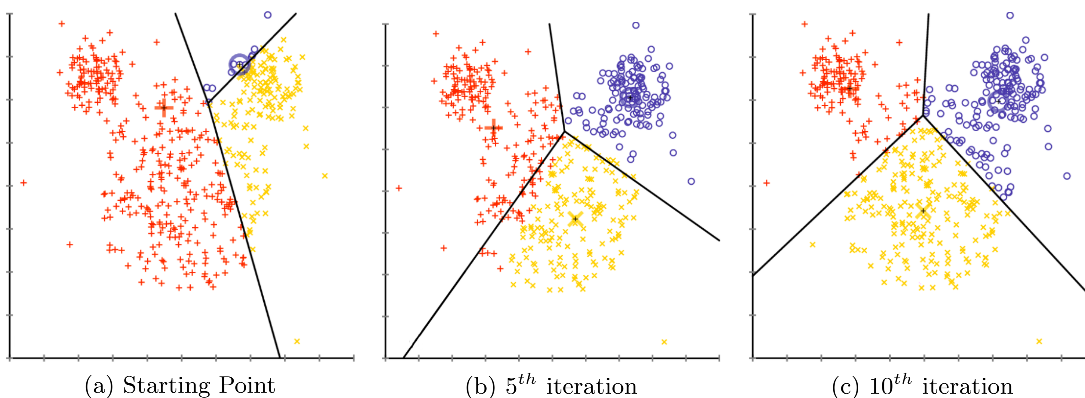
The distance used in this thesis is the Euclidian one because it tends to give better clustering performances ([55]).

[13]With $[\cdot]_d$ we indicate the dth element of the vector inside the brackets. This notation is used here to avoid misunderstandings, since the vector already presents a subscript.

The next step is the *updating* of all the K centroids, and this is done as follows:

$$c_k = \frac{1}{|S_k|} \sum_{x_i \in S_k} x_i \quad , \quad \forall k \tag{3.37}$$

Once this is done, the algorithm restart the process, reassigning all the samples to the K clusters in the same way as done in the first step, but using the new found centroids. It generally stops when no point is reassigned to a cluster that is different from the one it belonged at the step before. Computationally, this always leads to convergence. To grafically see an easy 2-dimensional example of how it works, see Figure 3.9.



(a) Starting Point (b) 5th iteration (c) 10th iteration

Figure 3.9: Example (taken from [5]) in 2 dimensions of K-means algorithm, with $K = 3$ and random initialization of the centroids. Note how the three centroids change position during the process.

There exist more than one way to initialize the clusters. The common one for *K-means* consists in randomly choosing K samples from the dataset and use them as first centroids. The algorithm doesn't always find the global optimum since the result may depend on the initialization, but thanks to the fact that WCSS is quadratic (and so positive), at least a local optimum is always reached.

3.2.2.2 Application to Images

Let $X = \{x_i, \, i = 1, ..., I\}$, with every $x_i \in \mathbb{R}^D$, be the local descriptors extracted from an image. As for the FV, the encoder could be applied to any subset of features $Y \subset X$, but we will keep the already used notation. The objective is to create a vocabulary $C = [c_1, ..., c_K] \in \mathbb{R}^{D \times K}$, also called *codebook* (this is equal to the set $\mathcal{S}$ of the notation used in 3.2.2.1), and to assign each local feature x_i to its closest *visual word*[14] in the dictionary. The *bag* of visual words corresponds to the vector of occurrence counts of the visual words, i.e. a histogram over the vocabulary C (see Figure 3.10).

[14]The use of the nomenclature "visual word" derives from the fact that this representation was originally used in *Natural Language Processing* (NLP) problems. In the context of image classification, the so-called *words* are represented by the image features.

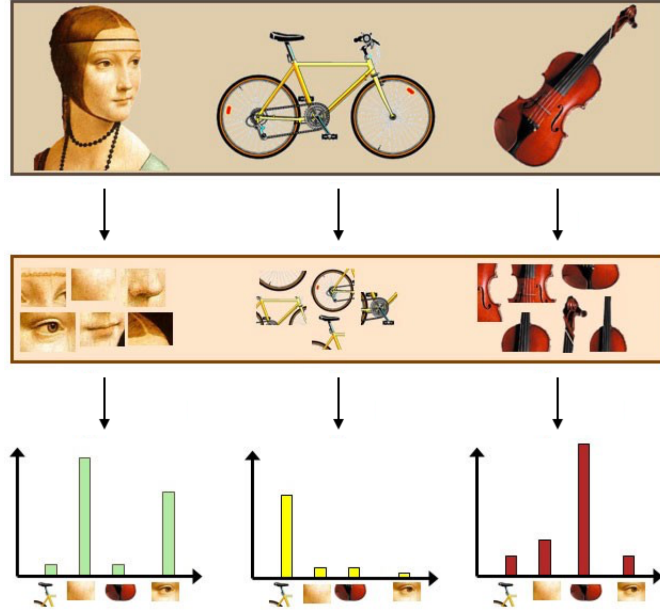


Figure 3.10: In this image (taken from [1]) it is shown how the BoVW works: first the features are extracted and then each image is represented as a frequency histogram of its features. Here the constructed vocabulary has $K = 4$.

These visual words are obtained during the training step by performing the *K-means* clustering algorithm over the local features x_i. The centroids c_k of the clusters corresponds to the visual words of the vocabulary and are randomly initialize. Mathematically speaking, the occurrence counts for each cluster are:

$$b_k = \#\{x_i\colon\ x_i \text{ is assigned to } c_k\} \tag{3.38}$$

The final BoVW of X is aggregated and average pooled as follows:

$$\mathcal{B}^X = \big(b_1, ..., b_K\big) \quad , \quad \mathcal{B}^X \leftarrow \frac{1}{I}\mathcal{B}^X \tag{3.39}$$

Besides this, all the techniques of *post-processing* normalization applied afterwards are the same as the ones described in 3.2.1.3.
Note that the BoVW algorithm assigns just a single visual word to each of the features descriptors in an image. This is called *hard-assignment* and it is one of the biggest differences between the BoVW and the FV encoder. This way to assign samples to clusters mainly presents two problems:

- Codeword uncertainty: problem of selecting the correct visual word out of two or more relevant candidates.
- Codeword plausibility: problem of selecting a visual word without a suitable candidate in the codebook.

Compared with the *soft-assignment* used for FV, the *hard-assignment* does not delivers the mean information of code words and the shape of their distribution. Even with all these weaknesses, the BoVW still represents the state-of-the-art in many applications, e.g. large scale content based image retrieval.
In practice it has been proven ([18]) that for the BoVW the average pooling could be substituted with the max pooling and that, depending on the problem and on the normalization used in the post-processing, this could lead to a better performance. Nevertheless, in this thesis we keep using the first described pooling in order to better compare the results from the different encoders.
The BoVW general advantages are the invariance to scale and orientation of the image and the good interpretability of the encoding, while the main disadvantages are that it ignores the spatial relationships among the local descriptors and that creating the codebook from large amount of data is generally costly.
The pseudo-code of the BoVW algorithm is shown in alg. 2. Here it is not reported its post-processing normalization, as for the FV, even if performed in practice.

Algorithm 2: Bag of Visual Words (BoVW) Algorithm

Input:
- Number of clusters: K
- Local Image Descriptors: $X = \{x_i\}_{i=1}^{I}$, $x_i \in \mathbb{R}^D \ \forall i$

Output:
- Bag of Visual Words: $\mathcal{B}^X \in \mathbb{R}^K$

1. **K-means Algorithm** - Creation of the Codebook:

   - Initialization of the clusters centroids $c_k \in \mathbb{R}^D$, $\forall k = 1, ..., K$

   while at least one x_i changes cluster **do**
   - Assign each of the x_i to its nearest c_k, using the *Euclidian Distance*
   - Update the centroids c_k of the cluster S_k:

   $$c_k = \frac{1}{|S_k|} \sum_{x_i \in S_k} x_i \quad , \quad \forall k$$

   end

2. **BoVW Aggregation**
   - $\forall k$, count how many features x_i are assigned to its corresponding cluster:

   $$b_k = \#\{x_i \colon \, x_i \text{ is assigned to } c_k\}$$

   - Aggregate all the b_k in the so-called BoVW of X: $\quad \mathcal{B}^X = \big(b_1, ..., b_K\big)$
   - Average Pooling of the BoVW: $\quad \mathcal{B}^X = \frac{1}{I}\mathcal{B}^X$

3.2.3 Vector of Locally-Aggregated Descriptors

The *Vector of Locally-Aggregated Descriptors* (VLAD) is a pooling encoder first introduced in 2010 by H. Jégou et al. ([29]).
The idea behind this technique is the same that exists behind the already explained FV, but joined with what we showed about the BoVW. Indeed, FV and VLAD are so similar that the differences between them could be resumed in these two observations:

- Like in BoVW, the local image descriptors are first assigned to their corresponding visual word in the K elements vocabulary, using as *quantizer* the already described *K-means* algorithm (instead of using the GMM as in FV[15]).
  Indeed, the *soft-assignment* used for the FV (i.e. the posterior probability of each GMM component), is replaced with the following *hard-assignment*:

$$v_k = \sum_{x_i:\ x_i \text{ is assigned to } c_k} (x_i - c_k) \tag{3.40}$$

  where x_i is the input local descriptor and c_k its closest visual word in the vocabulary. The final VLAD representation of the image X, indicated with with $\mathcal{V}^X \in \mathbb{R}^{K \times D}$, is the concatenation of the accumulated vectors v_k (see [12]):

$$\mathcal{V}^X = \left(v_1, ..., v_K\right) \tag{3.41}$$

  Note that all the v_k have the same size, which is equal to the size of the used local features. Before passing it to the classifier or perform the descriptor normalization presented in 3.2.1.3, we divide it, as done for the FV and the BoVW, by the sample size: $\mathcal{V}^X \leftarrow \frac{1}{I}\mathcal{V}^X$ (this is equal to perform the *average pooling*).

- Considering the FV expressed as in (3.27), we could see VLAD algorithm as a simplification (or a specific case) of the FV, where the contribution of the 2nd statistic is not taken into account. This usually leads to a less precise representation of the local descriptors and, practically speaking, to a weaker classification performance than the one obtained using FV.

[15] This is what is generally done in practice, even if theoretically also the GMM could be an option.

Chapter 4

Proposed Solution

In this chapter it is presented in detail the solution we propose to solve the classification problem of lung cancer cells images (Figure 4.1).

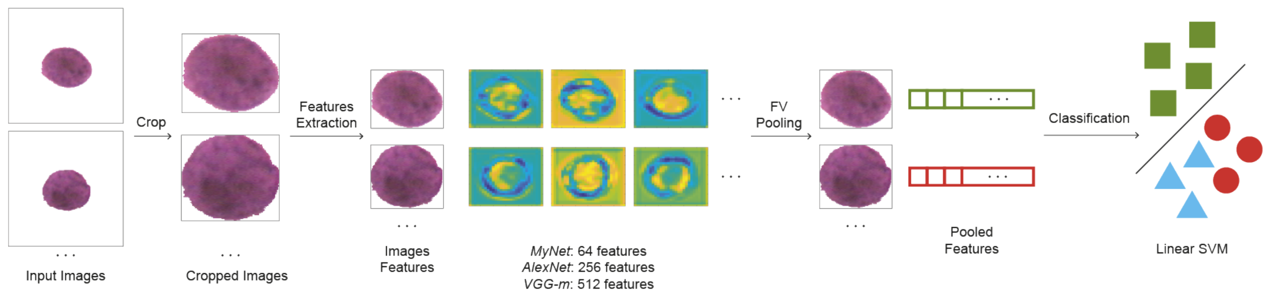


Figure 4.1: Detailed scheme of the proposed classification process. After cropping the images, the features are extracted. An encoder is then applied in order to obtain the vectors of pooled features (we write “FV Pooling” since it is the pooling encoder we propose to use). Finally, the vectors are passed to a linear SVM to be classified.

We first face the problem of extracting the features training the CNN from scratch. We constructed our own network, which from now on is called *MyNet*, taking mainly as example the *AlexNet* structure ([31]), but with some modifications. The main difference between them is that *MyNet* is shallower and with less parameters to be trained (*AlexNet*: ≈ 62 Millions, *MyNet*: ≈ 1.4 Millions). This is because the more parameters to train in a network, the more images are needed to obtain good classification results. In *MyNet* architecture, we also inserted two dropout layers between the fully-connected ones. This is done to partially avoid overfitting during the training step. Moreover, a cost, proportional to the number of samples in each class, is applied to each image output vector to compute the value of the loss function, which is chosen to be the so-called log-loss.

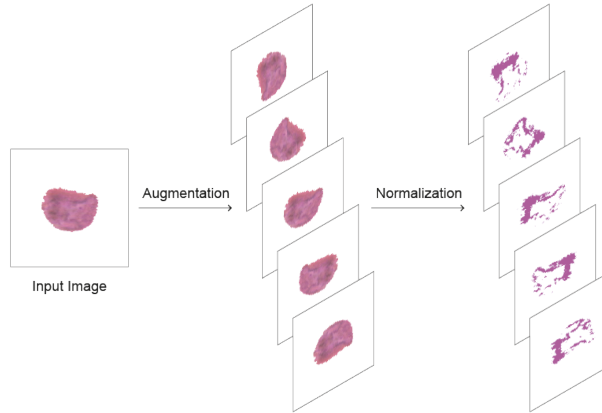


Figure 4.2: Images pre-processing when training *MyNet* from scratch and when fine-tuning *VGG-m* parameters. The first consists in augmenting the input image, the second one is the subtraction of the average training image.

The structure of *MyNet* is explained in details in table 4.1 and its training architecture is shown in Figure 4.3. Note that we set the input image size equal to 256×256, with 3 input channels (RGB format), to make the network more suitable for our dataset. Right before training, we subtract to each image the average of all the training images in the dataset (see Figure 4.2). This is done to normalize the input data and should be able to remove their "bias".

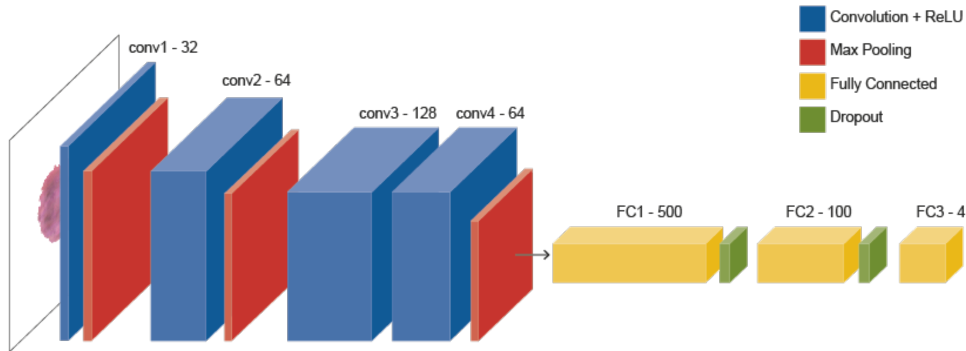


Figure 4.3: *MyNet* architecture used during training.

Once *MyNet* is trained, we use it to extract features. Before doing that, we crop the input images, circumscribing a rectangle around each cell, in order to remove the useless white background. Then, the features are extracted from the 9th layer (4th convolutional one) of the network and hence, for each image, we obtain 64 feature maps. Once the features are extracted, the three different pooling encoders are applied one at a time and their outcome is then used to separately train the linear SVM classifier. The whole process (see Figure 4.1), starting from the training of the network, is then performed again on the augmented dataset. Due to the scarce amount of data, the described procedure returns poor results and so we try to use a different approach.

The local descriptors are then obtained using transfer learning. This should allow the network to extract good features even without a great amount of data. In our work we do that exploiting two different pre-trained networks: the already mentioned *AlexNet* and *VGG-m* ([17]). These two networks are not very different between each other, both in terms of depth (5 convolutional layers and 3 fully-connected ones) and types of layers, but *VGG-m* generates more features maps in the last convolutional layers, and hence presents slightly more parameters ($\approx$ 100 Millions). Both the networks were originally trained on *ImageNet*, an enormous visual database containing more than 14 millions images grouped in more or less 20 thousands non-overlapping classes. The structures of these two networks are respectively shown in 4.2 and 4.3. Again, before extracting the images features, we crop the images to remove useless information. For both networks, the features are extracted after the 13th layer (5th convolutional one), when we want to apply a pooling encoder, and after the 19th (before the last fully-connected layer) when no encoder is used. When extracting from the 9th layer using *AlexNet*, the feature maps are 256 for each image, while when using *VGG-m* network they are 512. These features, with respect to the ones extracted using *MyNet*, are more precise and detailed (see the difference between their feature maps in Chapter 5), since they are obtained from a deeper convolutional layer. The classification results obtained extracting features with transfer learning overtake the ones of the previous method, both with and without using data augmentation on the training set.

What we propose next is to improve again this results fine-tuning the parameters of the pre-trained networks. Note that the networks have to be adapted when re-trained during fine-tuning, changing the last fully-connected layer's output dimension in order to make it classify into 4 classes instead of 1000. This is not necessary when the pre-trained networks are used directly to extract features. Anyway, we decide to fine-tune just the *VGG-m* network since, as it has more parameters, we expect it to perform better than *AlexNet*. The training architecture of *VGG-m* is shown in Fig. 4.4, while we do not report the *AlexNet* one since its parameters are not fine-tuned.

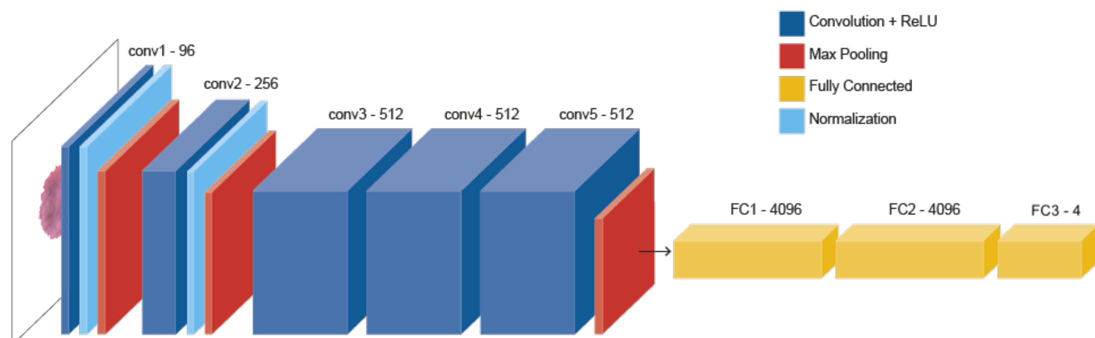


Figure 4.4: *VGG-m* architecture used during training. Note that the las fully connected layer outputs only 4 classes and not 1000, as it is in its original framework.

Also in this case, when calculating the loss, we decide to apply a cost to the images in order to reduce the imbalanced classes effect. We choose to optimize the parameters from the 9th to the last layer of the network, which in practice consists in adjusting the network weights of the last three convolutional layers and all the fully-connected ones, and freeze the first eight ones setting to 0 their parameters learning rate. This choice is purely empirical: we did not have enough computational power to fine-tune all the network layers' parameters, or even perform tests on how many layers would be ideal to adapt on our dataset. This is, in fact, something that we aim to discover in some further analysis. As we did with *MyNet*, before re-training the *VGG-m*, we remove the "bias" from the input images (see again Figure 4.2) subtracting the average image of the original training set the network was trained on (i.e. *ImageNet*). We could have subtracted the average of our specific training set but, since it is significantly smaller than *ImageNet*, we choose not to use it. After re-training the network, the process of extraction, pooling and classification with the linear SVM is performed exactly as before (Fig. 4.1).

Layer	Type	Input Size	Output Size	Support	Stride	Pad
1	Convolution	$256 \times 256 \times 3$	$63 \times 63 \times 32$	8×8	4	0
2	ReLU	–	–	1×1	1	0
3	Max-Pool	$63 \times 63 \times 32$	$31 \times 31 \times 32$	3×3	2	0
4	Convolution	$31 \times 31 \times 32$	$27 \times 27 \times 64$	5×5	1	0
5	ReLU	–	–	1×1	1	0
6	Max-Pool	$27 \times 27 \times 64$	$13 \times 13 \times 64$	3×3	2	0
7	Convolution	$13 \times 13 \times 64$	$13 \times 13 \times 128$	3×3	1	1
8	ReLU	–	–	1×1	1	0
9	Convolution	$13 \times 13 \times 128$	$13 \times 13 \times 64$	3×3	1	1
10	ReLU	–	–	1×1	1	0
11	Max-Pool	$13 \times 13 \times 64$	$6 \times 6 \times 64$	3×3	2	0
12	Fully-Connected	$6 \times 6 \times 64$	$1 \times 1 \times 500$	6×6	1	0
13	ReLU	–	–	1×1	1	0
14	Dropout	–	–	1×1	1	0
15	Fully-Connected	$1 \times 1 \times 500$	$1 \times 1 \times 100$	1×1	1	0
16	ReLU	–	–	1×1	1	0
17	Dropout	–	–	1×1	1	0
18	Fully-Connected	$1 \times 1 \times 100$	$1 \times 1 \times 4$	1×1	1	0
19	Softmax Loss	–	–	1×1	1	0

Table 4.1: *MyNet* architecture. The separation line between the 9th and 10th layers indicates where we extract the image features before applying the encoders.

Layer	Type	Input Size	Output Size	Support	Stride	Pad
1	Convolution	$227 \times 227 \times 3$	$55 \times 55 \times 96$	11×11	4	0
2	ReLU	–	–	1×1	1	0
3	Normalization	–	–	1×1	1	0
4	Max-Pool	$55 \times 55 \times 96$	$27 \times 27 \times 96$	3×3	2	0
5	Convolution	$27 \times 27 \times 96$	$27 \times 27 \times 256$	5×5	1	2
6	ReLU	–	–	1×1	1	0
7	Normalization	–	–	1×1	1	0
8	Max-Pool	$27 \times 27 \times 256$	$13 \times 13 \times 256$	3×3	2	0
9	Convolution	$13 \times 13 \times 256$	$13 \times 13 \times 384$	3×3	1	1
10	ReLU	–	–	1×1	1	0
11	Convolution	$13 \times 13 \times 384$	$13 \times 13 \times 384$	3×3	1	1
12	ReLU	–	–	1×1	1	0
13	Convolution	$13 \times 13 \times 384$	$13 \times 13 \times 256$	3×3	1	1
14	ReLU	–	–	1×1	1	0
15	Max-Pool	$13 \times 13 \times 256$	$6 \times 6 \times 256$	3×3	2	0
16	Fully-Connected	$6 \times 6 \times 256$	$1 \times 1 \times 4096$	6×6	1	0
17	ReLU	–	–	1×1	1	0
18	Fully-Connected	$1 \times 1 \times 4096$	$1 \times 1 \times 4096$	1×1	1	0
19	ReLU	–	–	1×1	1	0
20	Fully-Connected	$1 \times 1 \times 4096$	$1 \times 1 \times 1000$	1×1	1	0
21	Softmax Loss	–	–	1×1	1	0

Table 4.2: *AlexNet* architecture. The separation line between the 13th and 14th layers indicates where we extract the image features before applying the encoders.

Layer	Type	Input Size	Output Size	Support	Stride	Pad
1	Convolution	$224 \times 224 \times 3$	$109 \times 109 \times 96$	7×7	2	0
2	ReLU	–	–	1×1	1	0
3	Normalization	–	–	1×1	1	0
4	Max-Pool	$109 \times 109 \times 96$	$54 \times 54 \times 96$	3×3	2	0
5	Convolution	$54 \times 54 \times 96$	$26 \times 26 \times 256$	5×5	2	1
6	ReLU	–	–	1×1	1	0
7	Normalization	–	–	1×1	1	0
8	Max-Pool	$26 \times 26 \times 256$	$13 \times 13 \times 256$	3×3	2	0
9	Convolution	$13 \times 13 \times 256$	$13 \times 13 \times 512$	3×3	1	1
10	ReLU	–	–	1×1	1	0
11	Convolution	$13 \times 13 \times 512$	$13 \times 13 \times 512$	3×3	1	1
12	ReLU	–	–	1×1	1	0
13	Convolution	$13 \times 13 \times 512$	$13 \times 13 \times 512$	3×3	1	1
14	ReLU	–	–	1×1	1	0
15	Max-Pool	$13 \times 13 \times 512$	$6 \times 6 \times 512$	3×3	2	0
16	Fully-Connected	$6 \times 6 \times 512$	$1 \times 1 \times 4096$	6×6	1	0
17	ReLU	–	–	1×1	1	0
18	Fully-Connected	$1 \times 1 \times 4096$	$1 \times 1 \times 4096$	1×1	1	0
19	ReLU	–	–	1×1	1	0
20	Fully-Connected	$1 \times 1 \times 4096$	$1 \times 1 \times 1000$	1×1	1	0
21	Softmax Loss	–	–	1×1	1	0

Table 4.3: *VGG-m* architecture. The separation line between the 13th and 14th layers indicates where we extract the image features before applying the encoders.

Chapter 5

Experiments

In this chapter, other than presenting the dataset, we show the classification results obtained applying to our set of images the learning techniques as explained in Chapter 4. The same experiment is carried out more than one time, changing just the *train* and *test* set, in order to have a more general point of view on our algorithm performances and to see how much they depend on the images chosen for training. For the sake of simplicity, we just reported the best results for each experiment. The code[1] is implemented using a MATLAB toolbox called *MatConvNet* (see [61]). This collection of functions develops a clever way to use convolutional neural networks, pooling encoders and many classifiers like SVM. In the world of computer vision (in particular image understanding) this toolbox essentially exploit the open source library VLFeat ([60]).

5.1 Dataset

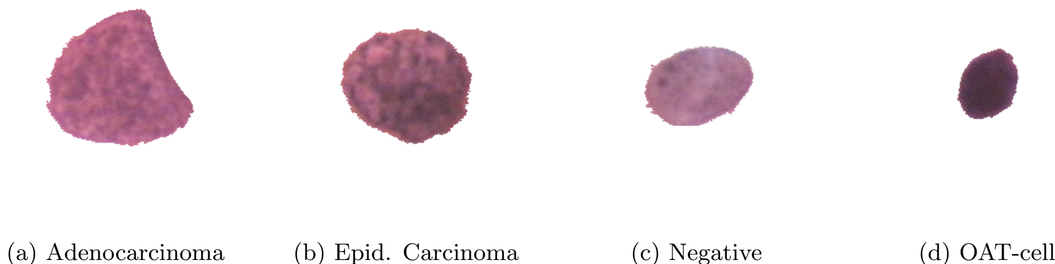
The dataset[2] consists of 5277 lung cells images divided in four imbalanced classes:

- Adenocarcinoma (529 images): *Non-Small Cell Lung Cancer* (NSCLC), usually occur at the outer edges of the lung.
- Epidermoid Carcinoma (484 images): *Non-Small Cell Lung Cancer* (NSCLC), they are more prevalent in men and arises in the lining of the large air passageways, or bronchi.
- Negative (692 images): cells where the tumor is not present.
- OAT-cell Carcinoma (3572 images): *Small Cell Lung Cancer* (SCLC), smaller than normal cells, they barely present any cytoplasm. This is the more aggressive type of lung cancer; compared with NSCLC, OAT-cells have a shorter doubling time, higher growth fraction, and earlier development of metastases.

[1] The "skeleton" of the code takes a page out from the already cited article "Deep Filter Banks for Texture Recognition, Description and Segmentation" ([19]).

[2] Kindly offered by the professor Konradin Metze and his research group, Department of Pathological Anatomy, Faculty of Medical Sciences, University of Campinas, Brazil.

One example for each type of cell is showed in Figure 5.1.
The images are retrieved by digitalizing brush cytologic preparations obtained from 132 untreated patients. This is done using the *Hematoxylin and Eosin* (*H&E*) stain, one of the techniques most widely used in histology to make diagnosis. The resulting images are collected with a spatial resolution of 0.1 $\mu m/pixel$ (1.25 numerical aperture, 100 $\times$ oil immersion), as explained in [7]. Tumor and normal epithelial cells were identified independently by two trained pathologists and only the ones with concordant diagnosis were included in this study. The final result consists in 256×256, RGB, '*.bmp*' format images with luminance levels ranging between 0 and 255.



(a) Adenocarcinoma (b) Epid. Carcinoma (c) Negative (d) OAT-cell

Figure 5.1: Examples of lung cells for each class in the dataset.

To divide the *train* and *test* set, we assigned to every image an integer extracted from a *Discrete Uniform Distribution* between 1, 2 and 3. If the number assigned is equal to 3 we set the corresponding image to the *test* set, otherwise to the *train* one. Since in this work we use different versions of the dataset, we give now their description and the respective notations adopted:

- "*lung*" dataset: it is the already described dataset, i.e. the original one.
- "*aug_lung*" dataset: to create this dataset, an *offline data augmentation* is performed. In particular, for each lung cell image in the training set we create 5 more augmented images applying a random rotation in the angles range of $[-\frac{\pi}{2}, +\frac{\pi}{2}]$ and a random reflection (performed with probability 0.5) both in the horizontal and vertical direction. Some transformations, like rescaling or gaussian noise, are not taken into account because of the cells structure, and others, like shifting, because they wouldn't add any useful information. We choose to not augment the test set.
- "*ad-vs-ep_lung*" dataset: this dataset is made up of only two classes, *Adenocarcinoma* and *Epidermoid Carcinoma*.
- "*oat-vs-all_lung*" dataset: this dataset is constructed in order to have just two classes. The first is the *OAT-cells* one and the second is the union of all images in the other three classes.

5.2 Experiment 01

Here are analyzed all the described techniques on both the original dataset "*lung*" and the augmented dataset "*aug_lung*".

Training from Scratch

We start showing in Figure 5.2 the behaviour of the *loss function* with respect to the training epochs (just for "*aug_lung*" dataset, since it's the most significative one). The results are obtained using a batch size of 256 and a learning rate which decreases logarithmically with the epochs between 10^{-4} and 10^{-6}.

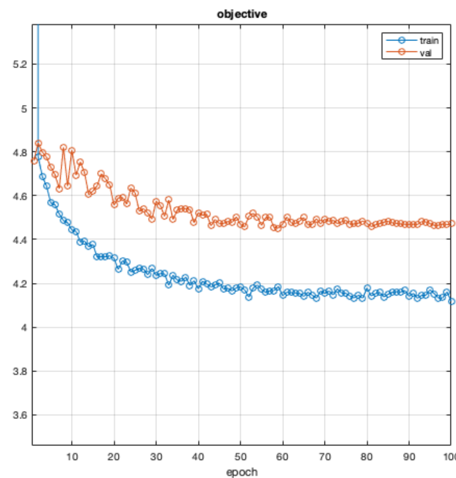


Figure 5.2: *Loss Function*: log-loss, the value continues to decrease but without any evident improvement.

Even if it's not very significative, since we will then use the SVM to finally classify the images, the behaviour of the loss function gives us information on the network ability to learn the important features of our images. After 100 epochs the objective is still decreasing, but the improvement is so slow that we decide to stop. Once *MyNet* is trained, we use it to extract the images features, which are then pooled and used to train the linear SVM classifier. To have an idea of how the extracted features look like, see one of the corresponding feature maps of a generic cell in Figure 5.3.

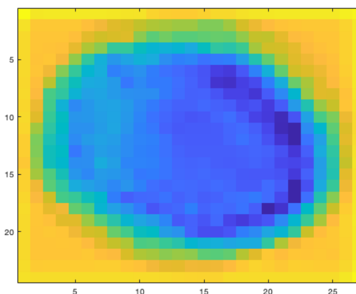


Figure 5.3: Feature map of a random chosen cell extracted after 9th layer of *MyNet*.

The obtained performances are shown in table 5.1, where we set 64 clusters for VLAD and FV, and 4096 for BovW. To respectively obtain the GMM parameters for FV and the visual words vocabulary for BoVW and VLAD, we use a subset of 1000 training images.

	BovW		VLAD		FV	
	Train	Test	Train	Test	Train	Test
lung	91.9	50.6	36.5	28.7	95.0	56.5
aug_lung	72.7	54.9	77.0	60.6	82.7	**64.0**

Table 5.1: Train and test mAP values (in %) when extracting features with *MyNet*.

We do not present the final results of the classification using no pooling encoder on the extracted features because the SVM was completely not able to interpret them and hence to correctly classify the images. The test performance of the algorithm improves in all the three cases applying data augmentation. The decreasing of the training mAP is instead a sign that we are succeeding in reducing overfitting. The results obtained using BoVW are acceptable, even if not impressive, while the ones obtained with VLAD are somewhat significative only with respect to the augmented dataset, but actually forgivable on the original one. The best classification results are obtained using "*aug_lung*" and FV as encoder (64.0%). Concerning this case, for completeness we also show in Figure 5.4 the APs for each class and in Figure 5.5 the respective train and test confusion matrices (which would not be significative without the AP values since the dataset is imbalanced).

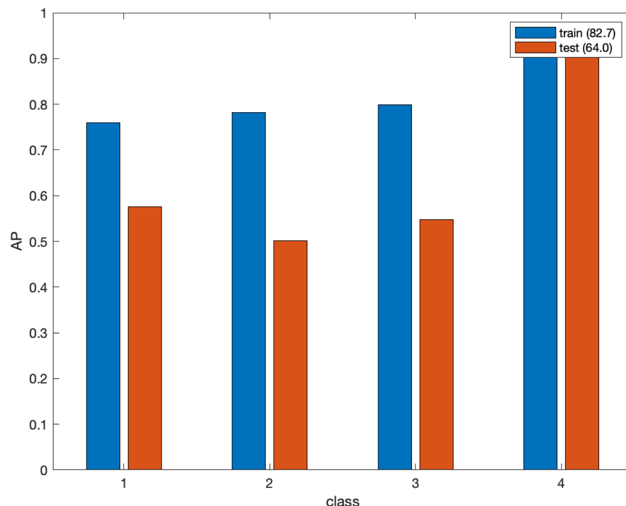
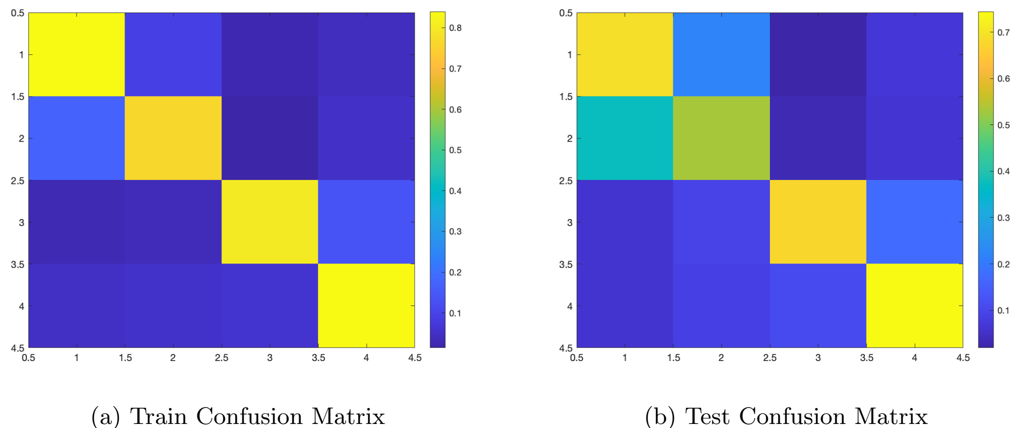


Figure 5.4: *MyNet* trained from scratch. APs of the four classes using FV encoder.

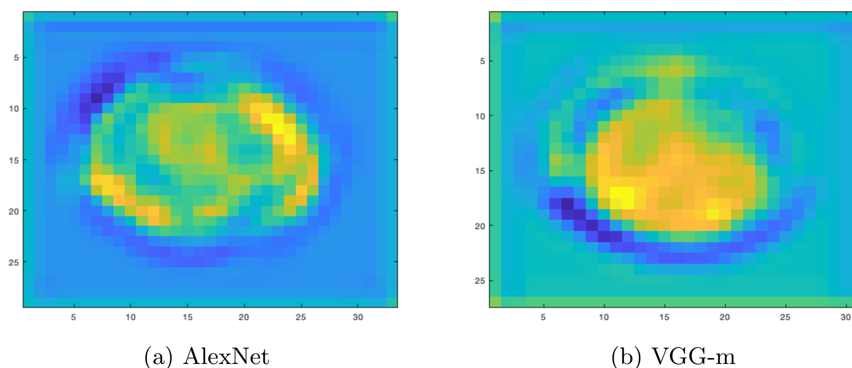


(a) Train Confusion Matrix (b) Test Confusion Matrix

Figure 5.5: 5.5a: total acc = 81.1%. 5.5b: total acc = 66.0%.

Transfer Learning

Here, to extract the important features from the input images, we use both *AlexNet* and *VGG-m*. See in Figure 5.6 an example of the feature maps for both networks. Looking carefully at the feature maps, it is possible to see that they present more detailed patterns than the one shown in Figure 5.3, which makes us expect a higher classification performance since they can identify more precise patterns in the images. This happens because the features extracted from the two pre-trained networks are obtained from a deeper layer than the one used to extract them from *MyNet*.



(a) AlexNet (b) VGG-m

Figure 5.6: Feature maps of a random chosen cell extracted after 13th layer of *AlexNet* (5.6a) and *VGG-m* (5.6b).

The hyperparameters used in this experiment framework are the same as for the network trained from scratch: 1000 images respectively to train and create the 64 GMM parameters for FV and the 4096 visual words vocabulary for BoVW and VLAD. The classification results obtained with the two pre-trained networks are showed in 5.2 when using *AlexNet* and in 5.3 when using *VGG-m*.

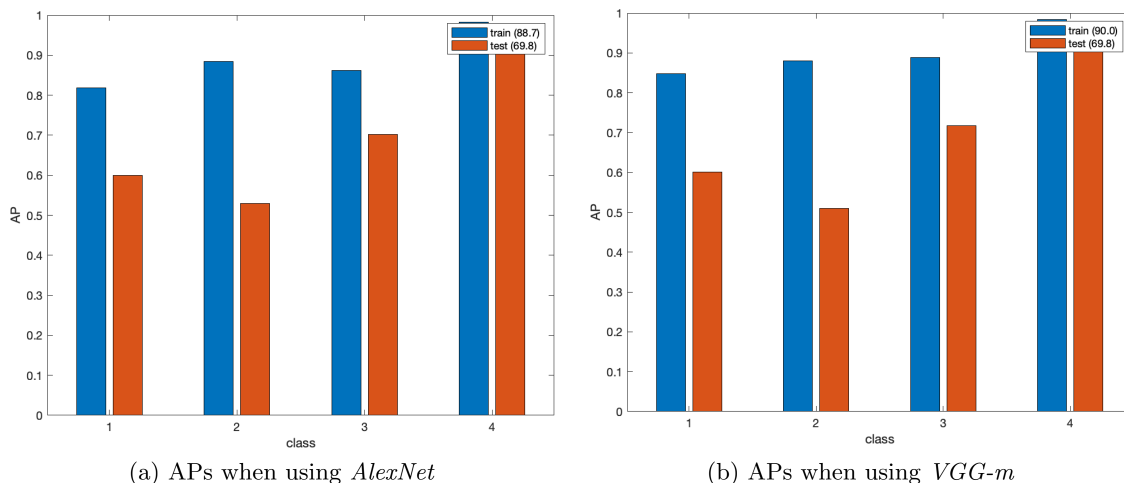
	No Enc.		BovW		VLAD		FV	
	Train	Test	Train	Test	Train	Test	Train	Test
lung	78.1	62.8	86.9	56.3	90.1	64.5	92.6	64.9
aug_lung	72.6	63.3	74.7	58.7	86.0	69.4	88.7	**69.8**

Table 5.2: Train and test mAP values (in %) when extracting features with *AlexNet*.

	No Enc.		BovW		VLAD		FV	
	Train	Test	Train	Test	Train	Test	Train	Test
lung	76.4	62.9	88.8	56.3	93.1	63.7	95.0	65.4
aug_lung	72.0	65.6	76.2	58.7	87.8	68.5	90.0	**69.8**

Table 5.3: Train and test mAP values (in %) when extracting features with *VGG-m*.

In all the above cases, the features are extracted at the 13th layer of the networks, except for the case where no encoder is used, in which they are extracted at the 19th one, right before the last fully-connected layer. In this scenario, the first two fully-connected layers of the network are interpretable as a pooling encoder that transforms the features before passing them to the linear SVM. The results are very similar between the two networks and the differences could actually be due to random choices, like the 1000 images used for training the GMM parameters and the visual words vocabulary. Anyway, transfer learning overtakes training from scratch, both with and without data augmentation. The difference is that the performance of the network trained from scratch improves much more with data augmentation than the pre-trained networks do because transfer learning already helps reducing overfitting. The FV with data augmentation remains the encoder giving the best performances (*AlexNet*: 69.6%, *VGG-m*: 69.8%). We show, for this last case, the resulting APs for both networks in Figure 5.7.



(a) APs when using *AlexNet*

(b) APs when using *VGG-m*

Figure 5.7: APs of the four classes using FV.

Even if we already reached some good results, we try to improve them *fine-tuning* the parameters of the pre-trained networks. Since the two networks performances are very similar, we do that just for the *VGG-m*. We freeze the first 8 layers of the network, setting their parameters learning rate to 0, while we train just the other ones. Their learning rate is set to decrease with the epochs from 10^{-3} to 10^{-4} and the batch size for the update with the gradient descent is set to 256. The behaviour of the loss function during the tuning of the parameters, when using the augmented dataset, is shown in Figure 5.8.

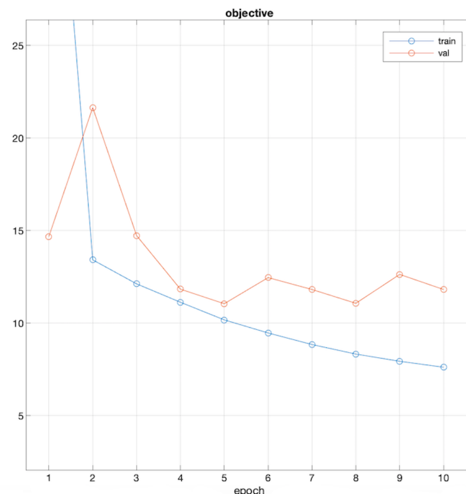


Figure 5.8: *Loss Function*: log-loss, the value continues to decrease for training, but stops and starts oscillating for test.

We fine-tuned the parameters for 10 epochs, but we stopped since after the 5th one it seems that the network starts overfitting. This is clear because the training value of the loss keeps decreasing while the test one starts growing back. Once the network is re-trained, both for the original and the augmented data, the classification is performed extracting features from the same layers and setting the same parameters as before. Since the FV already proved to be the highest performing encoder, we do not report the final mAPs for BovW and VLAD. We also show the classification results when no encoder is applied just to see if fine-tuning all the FC layers can, in some way, improve the performance of this classification structure (see table 5.4).

	No Enc.		FV	
	Train	Test	Train	Test
lung	62.9	51.6	94.9	63.6
aug_lung	69.9	63.5	89.8	**70.4**

Table 5.4: Train and test mAP values (in %) when extracting features with *VGG-m* fine-tuned from the 9th layer to the last one.

As expected, the performance obtained fine-tuning the *VGG-m* network is the best among all the others (70.4%). We show respectively in Figure 5.9 and Figure 5.10 the APs and the confusion matrices of this experiment. Unfortunately, the gain in mAP with respect to the standard transfer learning is exiguous. The fact is that the techniques used until now are already set to reduce a lot the overfitting of the network on training data and, because of that, the improvement due to fine-tuning is barely noticeable.

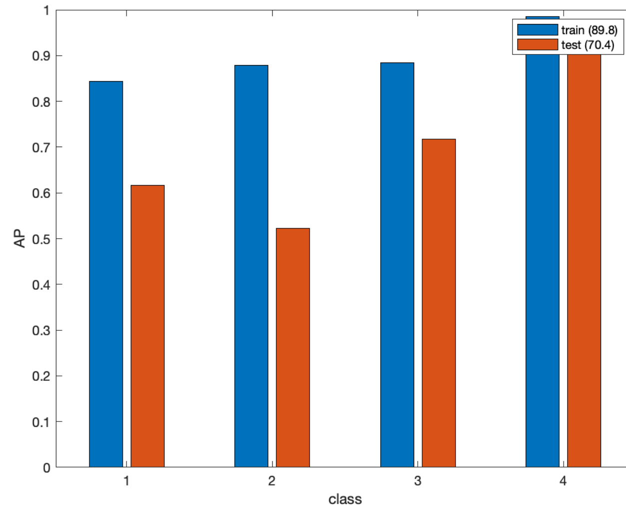
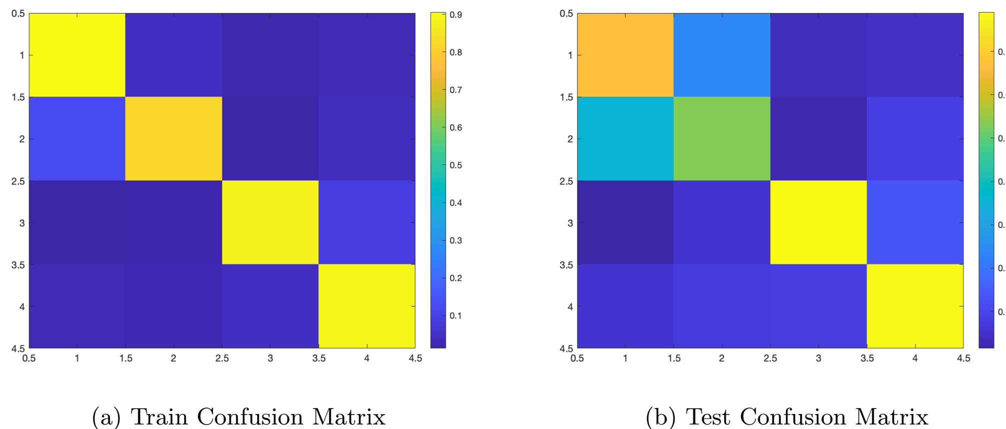


Figure 5.9: *VGG-m* fine-tuned. APs of the four classes using FV encoder.



(a) Train Confusion Matrix

(b) Test Confusion Matrix

Figure 5.10: 5.10a: total acc = 89.5%. 5.10b: total acc = 69.5%.

Something that is important to underline is that we put a lot of attention in parameters fine-tuning since the use of FV is computationally much more expansive than directly using fully-connected layers. This happens because the FV needs first to initialize and train the requested GMM parameters and, secondly, because its pooled

representation of the features is generally dense and hence, when working with big local descriptors, the computing time really represents a limit.

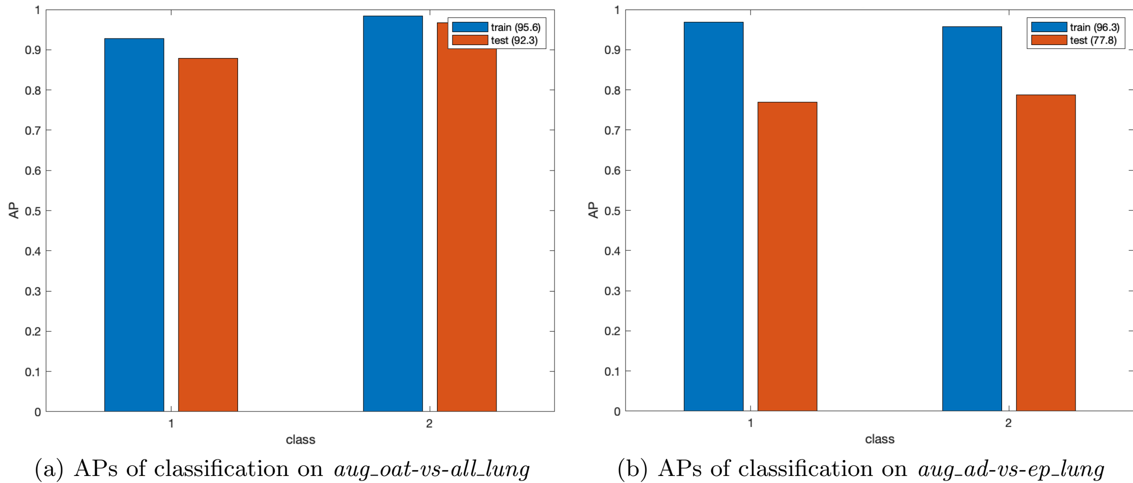
5.3 Experiment 02

This experiment does not aim to prove the efficiency of the proposed methods, but rather tries to highlight other doubts related to the dataset. The first one concerns the original reason why the imbalanced dataset. The class of *OAT-cells* contains much more images than other classes because, when the data was collected, the objective was finding an algorithm being able to differentiate between this class and all the others. This is why we created the "*oat-vs-all_lung*" dataset and we test our classification methods on it. The second reason derives from an analysis a posteriori on the outcomes we obtained in section 5.2. From the test APs of those experiments it was clear that the worst results of our classification algorithm were obtained when trying to detect the attributes of cells in the first two classes: *Adenocarcinoma* and *Epidermoid Carcinoma*. We identified the two main causes of this inability: (1) these are the two classes with less samples in the dataset, and hence our algorithm struggles to learn which are the main characteristics of these two types of cell; (2) they really are very similar in color, shape and especially size of the tumor (as explained in section 5.1, they both belong to the *Non-Small Cell Lung Cancer* family). Hence, we create the "*ad-vs-ep_lung*" dataset in order to test our classification algorithm just on these two classes.
This experiment is performed extracting image features with pre-trained *VGG-m* network without fine-tuning its parameters and using only the FV encoder, since it has already proven to be the best one. We also perform the classification algorithm applying data augmentation on both the described dataset as it was applied on "*lung*" in the first experiment. We call the two augmented dataset "*aug_oat-vs-all_lung*" and "*aug_ad-vs-ep_lung*", respectively. As before, we use a subset of 1000 training images to create the 64 GMM parameters[3]. The results are shown in tab. 5.5.

	FV	
	Train	Test
oat-vs-all_lung	96.7	89.9
aug_oat-vs-all_lung	95.6	**92.3**
ad-vs-ep_lung	96.9	72.0
aug_ad-vs-ep_lung	96.3	**77.8**

Table 5.5: Train and test mAP values (in %) when extracting features with *VGG-m*.

The results are better than expected. Note that, as it happened before, the two experiments are improved by applying data augmentation, both reducing overfitting and increasing the test mAP (*aug_oat-vs-all_lung* mAP: 92.3%, *aug_ad-vs-ep_lung* mAP: 77.8%). The APs for the classification on the two augmented dataset are shown in Figure 5.11.



(a) APs of classification on *aug_oat-vs-all_lung* (b) APs of classification on *aug_ad-vs-ep_lung*

Figure 5.11: APs for the two augmented dataset using FV.

Looking at the results on the first dataset we can say that our algorithm is almost completely capable of identifying the differences between the two classes and hence to properly classify them. This is probably due to the fact that, treating as a single class the three low-represented ones, we partially reduce the imbalanced classes effect. Moreover, joining these three classes we are implicitly putting together the images most difficult to distinguish. Concerning the second set of data, the results still do not reach outstanding performances, always because the scarce amount of data, but at least they are acceptable considering the fact that the two classes in the dataset are the most similar and, as a consequence, the most difficult to correctly classify.

[3] Actually, when we are classifying "*ad-vs-ep_lung*", the total images are 1013. Since the training and the test images are assigned as explained in 5.1, the training set will contain less than 1000 images. This is the only case where to train the GMM parameters it is used the whole training set and not some randomly selected subset.

Conclusions

By performing classification of lung cancer cell images, using *Convolutional Neural Networks* as feature extractor and a linear *Support Vector Machine* as classifier, this work showed that the *Fisher Vector* pooling leads to excellent results, even better than using none or other types of state-of-the-art encoders. As expected, the transfer learning approach quite overtook the trained from scratch network's results. The performance of this framework has been improved by applying *data augmentation* at first, and *fine-tuning* the parameters of the pre-trained network then. Even if we saw a slight improvement in the results using these two techniques, we did not succeed in completely overcome the scarce amount of collected data, which indeed represents the biggest limit in this research. Despite this, our research was primarily able to prove the FV's ability of pooling the features extracted from lung cancer cell images in order to make them much more informative for a linear SVM classifier and, secondly, that applying transfer learning with data augmentation and parameters fine-tuning really helps to reduce overfitting. In particular, we think that parameters fine-tuning is an efficient technique to apply when using transfer learning and the target dataset is very different from the one used to originally train the network. Because of this, we think that this technique should be taken into account for further works. Besides that, we believe that the presented learning structure can be considered a valid option in solving medical image classification tasks when a huge amount of training data is not available.

Bibliography

[1] https://towardsdatascience.com/bag-of-visual-words-in-a-nutshell-9ceea97ce0fb.

[2] https://en.wikipedia.org/wiki/Cluster_analysis.

[3] https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/.

[4] https://ec.europa.eu/eurostat/web/products-eurostat-news/-/EDN-20180531-1.

[5] https://en.wikipedia.org/wiki/K-means_clustering.

[6] https://cs231n.github.io/convolutional-networks/.

[7] R. L. Adam, R. C. Silva, F. G. Pereira, N. J. Leite, I. Lorand-Metze, and K. Metze. The fractal dimension of nuclear chromatin as a prognostic factor in acute precursor b lymphoblastic leukemia. *Analytical Cellular Pathology*, 28(1, 2):55–59, 2006.

[8] A. F. Agarap. An architecture combining convolutional neural network (cnn) and support vector machine (svm) for image classification. *arXiv preprint arXiv:1712.03541*, 2017.

[9] A. Aggarwal, G. Lewison, S. Idir, M. Peters, C. Aldige, W. Boerckel, P. Boyle, E. L. Trimble, P. Roe, T. Sethi, et al. The state of lung cancer research: a global analysis. *Journal of Thoracic Oncology*, 11(7):1040–1050, 2016.

[10] E. Ahn, A. Kumar, D. Feng, M. Fulham, and J. Kim. Unsupervised feature learning with k-means and an ensemble of deep convolutional neural networks for medical image classification. *arXiv preprint arXiv:1906.03359*, 2019.

[11] S.-i. Amari and H. Nagaoka. *Methods of information geometry*, volume 191. American Mathematical Soc., 2007.

[12] G. Amato, P. Bolettieri, F. Falchi, and C. Gennaro. Large scale image retrieval using vector of locally aggregated descriptors. In *International Conference on Similarity Search and Applications*, pages 245–256. Springer, 2013.

[13] L. Bontemps, J. McDermott, N.-A. Le-Khac, et al. Collective anomaly detection based on long short-term memory recurrent neural networks. In *International Conference on Future Data and Security Engineering*, pages 141–152. Springer, 2016.

[14] R. Boţ, G. Kassay, and G. Wanka. Strong duality for generalized convex optimization problems. *Journal of Optimization Theory and Applications*, 127(1):45–70, 2005.

[15] K. Boyd, K. H. Eng, and C. D. Page. Area under the precision-recall curve: point estimates and confidence intervals. In *Joint European conference on machine learning and knowledge discovery in databases*, pages 451–466. Springer, 2013.

[16] M. A. Carreira-Perpinán. A review of mean-shift algorithms for clustering. *arXiv preprint arXiv:1503.00687*, 2015.

[17] K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman. Return of the devil in the details: Delving deep into convolutional nets. *arXiv preprint arXiv:1405.3531*, 2014.

[18] H. Chougrad, H. Zouaki, and O. Alheyane. Soft assignment vs hard assignment coding for bag of visual words. In *2015 10th International Conference on Intelligent Systems: Theories and Applications (SITA)*, pages 1–5. IEEE, 2015.

[19] M. Cimpoi, S. Maji, I. Kokkinos, and A. Vedaldi. Deep filter banks for texture recognition, description, and segmentation. *International Journal of Computer Vision*, 118(1):65–94, 2016.

[20] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996.

[21] R. Frank. The perceptron, a perceiving and recognizing automaton (project para). *Cornell Aeronautical Laboratory Report No. 85-460-1*, 1957.

[22] S. Geman, E. Bienenstock, and R. Doursat. Neural networks and the bias/variance dilemma. *Neural computation*, 4(1):1–58, 1992.

[23] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.

[24] T. Grel. Region of interest pooling explained. *deepsense. io*, 2017.

[25] M. A. Hanson. On sufficiency of the kuhn-tucker conditions. *Journal of Mathematical Analysis and Applications*, 80(2):545–550, 1981.

[26] R. G. Hart, H.-C. Diener, S. Yang, S. J. Connolly, L. Wallentin, P. A. Reilly, M. D. Ezekowitz, and S. Yusuf. Intracranial hemorrhage in atrial fibrillation patients

during anticoagulation with warfarin or dabigatran: the re-ly trial. *Stroke*, 43(6): 1511–1517, 2012.

[27] T. Jaakkola and D. Haussler. Exploiting generative models in discriminative classifiers. In *Advances in neural information processing systems*, pages 487–493, 1999.

[28] A. K. Jain, J. Mao, and K. M. Mohiuddin. Artificial neural networks: A tutorial. *Computer*, 29(3):31–44, 1996.

[29] H. Jégou, M. Douze, C. Schmid, and P. Pérez. Aggregating local descriptors into a compact image representation. In *2010 IEEE computer society conference on computer vision and pattern recognition*, pages 3304–3311. IEEE, 2010.

[30] K. C. Kiwiel. Convergence and efficiency of subgradient methods for quasiconvex minimization. *Mathematical programming*, 90(1):1–25, 2001.

[31] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[32] T. Learning. Convolutional neural network for visual recognition, 2017.

[33] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer, 2012.

[34] T. Leung and J. Malik. Representing and recognizing the visual appearance of materials using three-dimensional textons. *International journal of computer vision*, 43(1):29–44, 2001.

[35] Q. Li, W. Cai, X. Wang, Y. Zhou, D. D. Feng, and M. Chen. Medical image classification with convolutional neural network. In *2014 13th International Conference on Control Automation Robotics & Vision (ICARCV)*, pages 844–848. IEEE, 2014.

[36] S. Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137, 1982.

[37] R. Luengo-Fernandez, J. Leal, A. Gray, and R. Sullivan. Economic burden of cancer across the european union: a population-based cost analysis. *The lancet oncology*, 14(12):1165–1174, 2013.

[38] Lung Cancer Europe (LuCE). Challenges in lung cancer in europe, 2016. https://www.lungcancereurope.eu/wp-content/uploads/2017/10/LuCE-Report-final.pdf.

[39] L. Medsker and L. C. Jain. *Recurrent neural networks: design and applications*. CRC press, 1999.

[40] A. Mikołajczyk and M. Grochowski. Data augmentation for improving deep learning in image classification problem. In *2018 international interdisciplinary PhD workshop (IIPhDW)*, pages 117–122. IEEE, 2018.

[41] M. Mohri, A. Rostamizadeh, and A. Talwalkar. *Foundations of machine learning*. MIT press, 2018.

[42] H. Mohsen, E.-S. A. El-Dahshan, E.-S. M. El-Horbaty, and A.-B. M. Salem. Classification using deep learning neural networks for brain tumors. *Future Computing and Informatics Journal*, 3(1):68–71, 2018.

[43] R. Moskovitch, D. Stopel, C. Feher, N. Nissim, and Y. Elovici. Unknown malcode detection via text categorization and the imbalance problem. In *2008 IEEE International Conference on Intelligence and Security Informatics*, pages 156–161. IEEE, 2008.

[44] F. Perronnin and D. Larlus. Fisher vectors meet neural networks: A hybrid classification architecture. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3743–3752, 2015.

[45] F. Perronnin, J. Sánchez, and T. Mensink. Improving the fisher kernel for large-scale image classification. In *European conference on computer vision*, pages 143–156. Springer, 2010.

[46] J. Platt et al. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. *Advances in large margin classifiers*, 10(3): 61–74, 1999.

[47] X. Qiao and Y. Liu. Adaptive weighted learning for unbalanced multicategory classification. *Biometrics*, 65(1):159–168, 2009.

[48] J. Ramírez, J. Górriz, F. Segovia, R. Chaves, D. Salas-Gonzalez, M. López, I. Álvarez, and P. Padilla. Computer aided diagnosis system for the alzheimer's disease based on partial least squares and random forest spect image classification. *Neuroscience letters*, 472(2):99–103, 2010.

[49] R. Rouhi, M. Jafari, S. Kasaei, and P. Keshavarzian. Benign and malignant breast tumors classification based on region growing and cnn segmentation. *Expert Systems with Applications*, 42(3):990–1002, 2015.

[50] S. Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[51] T. Saito and M. Rehmsmeier. The precision-recall plot is more informative than the roc plot when evaluating binary classifiers on imbalanced datasets. *PloS one*, 10(3):e0118432, 2015.

[52] J. Sánchez, F. Perronnin, T. Mensink, and J. Verbeek. Image classification with the fisher vector: Theory and practice. *International journal of computer vision*, 105(3):222–245, 2013.

[53] D. Scherer, A. Müller, and S. Behnke. Evaluation of pooling operations in convolutional architectures for object recognition. In *International conference on artificial neural networks*, pages 92–101. Springer, 2010.

[54] S. Shalev-Shwartz and T. Zhang. Stochastic dual coordinate ascent methods for regularized loss minimization. *Journal of Machine Learning Research*, 14(Feb): 567–599, 2013.

[55] A. Singh, A. Yadav, and A. Rana. K-means with three different distance metrics. *International Journal of Computer Applications*, 67(10), 2013.

[56] H. Steinhaus. Sur la division des corp materiels en parties. *Bull. Acad. Polon. Sci*, 1(804):801, 1956.

[57] N. Tajbakhsh, J. Y. Shin, S. R. Gurudu, R. T. Hurst, C. B. Kendall, M. B. Gotway, and J. Liang. Convolutional neural networks for medical image analysis: Full training or fine tuning? *IEEE transactions on medical imaging*, 35(5):1299–1312, 2016.

[58] C. Tan, F. Sun, T. Kong, W. Zhang, C. Yang, and C. Liu. A survey on deep transfer learning. In *International conference on artificial neural networks*, pages 270–279. Springer, 2018.

[59] W. D. Travis. Update on small cell carcinoma and its differentiation from squamous cell carcinoma and other non-small cell carcinomas. *Modern Pathology*, 25(1):S18–S30, 2012.

[60] A. Vedaldi and B. Fulkerson. Vlfeat: An open and portable library of computer vision algorithms. In *Proceedings of the 18th ACM international conference on Multimedia*, pages 1469–1472, 2010.

[61] A. Vedaldi and K. Lenc. Matconvnet: Convolutional neural networks for matlab. In *Proceedings of the 23rd ACM international conference on Multimedia*, pages 689–692, 2015.

[62] I. I. Wistuba, D. Bryant, C. Behrens, S. Milchgrub, A. K. Virmani, R. Ashfaq, J. D. Minna, and A. F. Gazdar. Comparison of features of human lung cancer cell lines and their corresponding tumors. *Clinical cancer research*, 5(5):991–1000, 1999.

[63] B. Yegnanarayana. *Artificial neural networks.* PHI Learning Pvt. Ltd., 2009.

[64] H. Zen and H. Sak. Unidirectional long short-term memory recurrent neural network with recurrent output layer for low-latency speech synthesis. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4470–4474. IEEE, 2015.

[65] X. Zhang, Y. Zou, and W. Shi. Dilated convolution neural network with leakyrelu for environmental sound classification. In *2017 22nd International Conference on Digital Signal Processing (DSP)*, pages 1–5. IEEE, 2017.

[66] Y. Zhang and X. Xing. Classifications of hand and wrist fractures. In *Clinical Classification in Orthopaedics Trauma*, pages 183–230. Springer, 2018.

[67] Y. Zhang, Z. Dong, A. Liu, S. Wang, G. Ji, Z. Zhang, and J. Yang. Magnetic resonance brain image classification via stationary wavelet transform and generalized eigenvalue proximal support vector machine. *Journal of Medical Imaging and Health Informatics*, 5(7):1395–1403, 2015.

Ringraziamenti

Come accaduto spesso nella mia vita, mi sto nuovamente fermando per chiedermi chi sono ora e chi sarò in futuro. La risposta alla seconda domanda è nascosta nelle pieghe sconosciute del caso e dell'incertezza. Mi piace che sia così e che, fortunatamente, non sia oggetto di discussione delle mie prossime righe. La risposta alla prima, invece, è probabilmente frutto delle innumerevoli esperienze che ho vissuto e che, intersecandosi, mi hanno permesso di diventare la persona che oggi ho di fronte allo specchio. Raccontarle tutte sarebbe impossibile; è perciò mio obiettivo il tentare di applicare le mie scarse doti di sintesi per ringraziare chi, a queste esperienze, ha preso parte e ha contribuito alla mia crescita e conoscimento personale.

Vorrei cominciare ringraziando le due figure accademiche che mi hanno accompagnato e supportato negli ultimi mesi: il professor João Batista Florindo per avermi proposto questo incredibile progetto e per avermi accolto ed aiutato sin dal primo momento e il professor Giacomo Boracchi per aver visto in me del potenziale e essere stato sempre presente nonostante la distanza.

Procedo ringraziando chi nella mia vita è stato al mio fianco dall'inizio, indipendentemente dalle scelte fatte. Mamma, ho sempre ammirato la tua capacità di rialzarti pur andando tutto storto. Ti ringrazio per avermi mostrato come affrontare la vita di petto e con coraggio, per avermi insegnato l'importanza delle parole e anche delle lacrime. Papà, grazie per riuscire sempre a strapparmi un sorriso, anche nei momenti più difficili. L'amore e la gioia che emani sono inebrianti. Ti ringrazio anche per avermi trasmesso la tua passione per i viaggi, la tua curiosità e avermi insegnato come andare d'accordo con l'ansia. Pietro (Peter) per tutte le botte, gli abbracci e le rime che ci siamo scambiati, per avermi mostrato e insegnato cosa significa empatia e per essere diventato un amico, oltre che un fratello. Sei il mio esempio. Alla zia Virgi e allo zio Renny, per essere stati come dei genitori e per avermi insegnato che essere felici non è poi così difficile se si ha vicino chi si ama. Grazie anche per i funghi raccolti insieme e per i cenoni in taverna. Grazie agli zii Annalisa ed Ezio e ai miei cugini Sofia e Luca, per tutte le estati passate al Camping "La Rocca" e per esservi da sempre dimostrati dei veri affetti. Grazie anche a Gianluca, per essere stato al fianco di mamma, anche nelle avversità, e per avermi voluto bene in maniera spontanea.

Un grazie ai Libanesi. A Moaad, compagno di mille avventure dall'infanzia ad oggi, per le nostre letture, il Marocco e per essere stato la prima persona con cui ho sentito di poter essere completamente me stesso. Rappresenti un punto fermo nella mia vita. Al Cresh, per avermi sempre considerato come un fratello, per le tue storie inventate, le partite ai Pokemon a casa tua in Umbria e per avermi portato al mio primo vero concerto. Al Sambu, per avermi insegnato che diverso è bello, che certe cose accadono e basta e per avermi saputo far prendere il volo quando non riuscivo nemmeno a saltare. Ti stimo molto. Grazie anche a Tonio, Vargas, il Max, Josef e tutti gli altri per essere riusciti a rimanere uniti nonostante i tanti intoppi.

Un grazie alla Charly per avermi sempre sostenuto da lontano e per essere andata oltre le apparenze quando ci siamo conosciuti.

Grazie ai rugbisti. A Loris, educatore e mentore, grazie per avermi fatto capire che nulla è dovuto, per avermi insegnato la meritocrazia e il rispetto. Grazie ai Rumba. A Giordi, che per quanto io mi allontanassi dal nido, hai sempre trovato un modo per ricordarmi quanto è importante casa e quanto io sia sempre il benvenuto. A Doni, che sei passato da essere chi si occupava di me, a farti portare a casa ubriaco dopo i tornei. Sei più che un amico. A Destro, per esserti rivelato più che un ragazzo a cui piacciono i kebab, grazie per le risate e per essere fortissimo a Rugby08. Un grazie anche al Tumi, Zanna, Umbi, Carlo, Gimmy, Leo, Sic, Signo, Marcolongo e tutti gli altri per aver condiviso con me una parte così importante e impattante delle nostre vite.

Un grazie a Via Vallazze 91 e a GLS. Ad Aio (Lëtrë) per aver trasformato in Casa un luogo che non lo era, per aver condiviso risate e pianti, per il tuo sentimentalismo e per avermi sempre tenuto con lo stomaco pieno. Sei parte della mia famiglia. A Teo, per essere sempre riuscito a capirmi a pieno, anche nelle sfumature più incoerenti di me, per le prime barre, le tante birre e gli infiniti abbracci. Non mi stanchi mai. A Simo (Cos), per le strimpellate e i canti a squarciagola, per il nostro slang, le nostre chiacchierate lunghissime e per il tuo modo di amare. Sei una delle persone di cui mi fido di più. A Giodaz (Flaco), per i concerti e le nottate, per i passaggi in macchina e le numerose schimicate alle 3 di notte, grazie per la musica che abbiamo condiviso e per essermi sempre stato vicino rimanendo sempre tu. Ad Albi, grazie per avermi mostrato i miei difetti, per la tua gentilezza e per tutte le ore che, passate insieme a crescere, diventavano minuti. Vorrei tanto non averti mai deluso. A Leti, per i sorrisi, le occhiate di intesa e per avermi fatto di nuovo sentire un bimbo in un mondo dove per molti diventare adulti è un imperativo. Ad Andre per le mille paglie fumate, per i pasti condivisi a casa e per avermi fatto vedere che non è mai troppo tardi per esagerare. Ad Angelì, per i tuoi tanti consigli e le serate pazze, per le giornate di studio a tenermi "sul pezzo" e per esserti fidato di me dal primo momento che ci siamo incontrati. A Giopog, per Stranger Things, per essere stato sempre molto diretto, per non aver paura di dire la tua in situazioni scomode e per avermi insegnato ad essere auto ironico. A Jimmy per i nostri "workout", per essere così diversi ma volerci così bene e per averci portato a casa vivi dopo un'avventura di ritorno dalla Sardegna. Un grazie ad Apotz per gli

insulti amorevoli, ad Abba per i biscotti preparati insieme, a Dipa per le discussioni da Otaku e a Nene per i rientri insieme a casa di notte. Un grazie anche a Cami, Nderep, GrandpaEug, Lux, Lollo, Buch, Bea, Tina e Tower, avete reso la mia vita motivo di orgoglio.

Un grazie ai "Mate". Ad Abba (Kaldo), per le punchline pregiate e per le nottate passate ad ascoltare musica e parlare dei nostri problemi senza sosta. Ad Annina, per la tua parlantina, per i passaggi in macchina e per l'affetto sotto forma di insulti che non mi hai mai nascosto. A Magiu, per le cotolette che mi portavi, le partite di Macchiavelli e la fiducia che hai sempre dimostrato di avere in me. Un grazie anche a Manfra e JJ sottone, mi mancate davvero tanto.

Un grazie a Franco per avermi ricordato che la quantità di tempo trascorso con qualcuno non fa la qualità e a GiulioP per riuscire ancora a sorprendermi spostando ogni volta l'asticella della pazzia verso l'alto.

Un grazie al Patio e ai ragazzi di "freesta". A Uezzo, per le grasse risate e per avere sempre la risposta pronta. A Michi, per avermi trasmesso la passione che metti in quello che fai. A Gio, per la tua capacità organizzativa e la serenità che trasmetti. A Pit, per qualche storia e una serata indimenticabile. A Luke per i tuoi cappellini e i tuoi pareri musicali. Un grazie anche ad Albi C., Ale, Co, Nico e tutti gli altri. Grazie per tutti i freestyle interminabili e per le partite a scacchi.

Un grazie infine al Brasile e alla República dos franceses. A Nat per la tua forza di volontà, il tuo entusiasmo e per esserti trasformata in una sorella, a Fabi per essere la capa migliore che abbia mai conosciuto, a DZ per i nostri sguardi e i nostri silenzi, allo zio Vitti per le ore passate insieme ad ascoltare musica e lo scambio culturale creatosi, a Davinho per aver portato un po' di Italia e per esserti aperto e fidato di me, a Bruno per le discussioni infinite, a Gabs per le pazzie e lo smalto, a Billy per le tracce e le conversazioni sul terzo occhio. Un grazie anche a tutti quelli che non ho nominato, la mia vita e soprattutto io siamo cambiati e migliorati tanto dopo avervi conosciuto, vi porterò per sempre nel mio cuore, até mais.

Un altro capitolo della mia vita fa capolinea, uno nuovo prenderà il suo posto e molte cose cambieranno. Dovunque le circostanze mi condurranno, una parte della mia famiglia si troverà sempre dall'altra parte del Mondo, e per me questo è semplicemente un motivo per non fermarmi mai. Tante cose belle giungono così al termine. Dal canto mio, le abbraccio e le lascio con un sorriso, con la speranza che questo faccia da inizio per un ancor più felice futuro.

I cannot die because this is my Universe.
Non posso morire perché questo è il mio Universo.