Politecnico di Torino
Master of Science Degree in Mechatronic Engineering

Master Thesis

Deep One-class Classification: a Deep Learning-based method for people recognition applied to grayscale images

Supervisor:
prof. Marcello Chiaberge

Candidate:
Tiziana Lasala

October 23, 2020

Abstract

This thesis is aimed at developing an algorithm able to recognize people instances in grayscale pictures. This is a One-Class Classification problem where objects of a particular class are identified compared to all other possible ones. The *class person* is called *positive class* or *target class*, while other items are referred to be in the *negative class*, also called *alien class*.
The biggest challenge is represented by the variety of objects opposed to the target class, which does neither allow to model the external class in a univocal way, nor to have all possible cases inside the training set.
This problem cannot be solved using traditional techniques of binary and multi-class classifications, precisely because there are no pre-defined classes, so no labeled data different from target class objects are available. Examples of the class of interest are categorized just using instances of the same class.

The algorithm is built using the Deep One-class Classification (DOC), an innovative Deep Learning-based method targeting OCC problems in computer vision field, like novelty detection, anomaly detection and mobile active authentication.
This approach relies on the concept of *transfer learning*, since an external multi-class dataset from an unrelated task, called *reference dataset*, is employed to learn deep features characterizing the *person class*, in addition to the one-class target dataset.
A pre-trained Convolutional Neural Network is used in combination with two loss functions, *compactness loss* and *descriptiveness loss*, that make the variance of features extracted from the target dataset smaller and minimize the cross-entropy loss of the reference dataset.
The proposed approach is able to achieve good results in the Area Under Curve (AUC) of the Receiver Operating Characteristic curve.

The classification of people in photo or in video frames makes its way in different areas, like virtual reality, autonomous driving and video surveillance. The use of this algorithm in the latter allows to have big advantages over the user's privacy, since it works on the device at the tip of the net.

“Dicono che prima di entrare in mare
il fiume tremi di paura.

A guardare indietro
tutto il cammino che ha percorso,
i vertici, le montagne,
il lungo e tortuoso cammino
che ha aperto attraverso giungle e villaggi.

E vede di fronte a sé un oceano così grande
che a entrare in lui può solo
sparire per sempre.

Ma non c’è altro modo. [...]

Solo entrando nell’oceano
la paura diminuirà,
perché solo allora il fiume saprà
che non si tratta di scomparire nell’oceano
ma di diventare oceano.”

Khalil Gibran

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Objective of the thesis

This thesis is aimed at developing a Deep Learning algorithm able to recognize people instances in pictures.
This is a One-Class Classification problem where objects of a particular class are identified compared to all other possible ones. The *person class* is called *positive class* or *target class*, while other items are referred to be in the *negative class*, also called *alien class*.
The biggest challenge is represented by the variety of objects opposed to the target class, which does neither allow to model the external class in a univocal way, nor to have all possible cases inside the training set.
This problem cannot be solved using traditional techniques of binary and multi-class classifications, precisely because there are no pre-defined classes, so no labeled data different from target class objects are available.
Examples of the class of interest are categorized just using instances of the same class.

The algorithm is built using the Deep One-class Classification (DOC), an innovative Deep Learning-based method targeting OCC problems in computer vision field, like *novelty detection*, *anomaly detection* and *mobile active authentication*.
This approach relies on the concept of *transfer learning*, since an external multi-class dataset from an unrelated task, called *reference dataset*, is employed to correctly learn deep features characterizing the *person class*, in addition to the one-class target dataset.
A pre-trained Convolutional Neural Network, the high-performance MobileNetV2, is used in combination with two loss functions, *compactness loss* and *descriptiveness loss*, that make the variance of features extracted from the target dataset smaller and minimize the cross-entropy loss of the reference dataset.
In this way, we achieve specialized features for Deep One-class Classification: they have the two fundamental properties of *compactness*, which means that objects of the same category have similar features located close to each other, and *descriptiveness*, that is elements of different categories have different features placed apart from the rest.

The training part is carried out using *grayscale images*, obtained from RGB pictures of *Open Images Dataset V4* and *ILSVRC 2012 dataset*, properly selected

and pre-processed.
This choice is motivated by a future extension of the Deep One-class Classification in InfraRed images, in order to recognize individuals in frames coming from surveillance videos, even at night.
The testing part is realized by a *template matching framework*, where, firstly, some baseline features of person intances are stored and, then, a score is generated considering the *Euclidean distance* between new and stored features.
There are three testing datasets, *BN1*, *BN2* and *IR*, the latter containing InfraRed pictures, and all models are evaluated using metrics: *precision*, *recall*, *F1 score*, *accuracy* and the *Area Under Curve* (AUC) of the ROC curve.
Also some graphical tools are employed to make comparisons among resulting methods, like *Receiver Operating Characteristic* (ROC) curves, *Detection Error Tradeoff* (DET) curves and the *t-distributed Stochastic Neighbor Embedding* (t-SNE) visualization of features.
The proposed approach is able to achieve very good results in all measurements, also compared to binary classification algorithms, which require instead a particular configuration of the training dataset.

1.2 Organization of the thesis

This work of thesis covers six chapters, whose content is presented below.

In *Chapter 1 - Introduction*, the main objective of the thesis is shown, with a brief overview of its general structure.

Chapter 2 - Machine Learning offers a basic understanding of the interesting world of Machine Learning and, later, explores Deep Learning tecniques used in the development of the Deep One-class Classification algorithm, like Artificial Neural Networks and Convolutional Neural Networks.

In *Chapter 3 - OCC State of Art*, firstly, an investigation into general concepts of classification is made, highlighting differences with other tasks like regression. Later, the classification of a single class is defined and explored in contrast to binary and multi-class classifications. Finally, various applications of One-Class Classification are presented, followed by a roundup of popular approaches used to realize them.

Chapter 4 - Deep One-class Classification presents the method used for the development of the people recognition algorithm, called *Deep One-class Classification*. First of all, some key concepts of the method are explained, in particular what feature learning means and what is, instead, the transfer learning. Then, common frameworks used in Deep Learning, including the used framework in DOC, are shown. Finally, the DOC optimization problem, the two misurable loss functions and the training architecture are presented.

In *Chapter 5 - Deep One-class Classification of people*, the current implementation of the algorithm based on Deep One-class Classification is explained. First of all, the work platform necessary to develop the model during all phases is described. Then, the strategy used to create an effective training dataset is presented, with the related steps of image processing. Later, cores of DOC are analysed: the employed network MobileNetV2, training and testing frameworks, detailed computations for loss functions. Finally, an overview on testing datasets and on performance measures used to evaluate all models are shown.

In *Chapter 6 - Results and conclusions*, all obtained results are shown through tables and plots, finally including conclusions and suggestions on future work.

Chapter 2

Machine Learning

In this work of thesis a *One Class Classification* algorithm is developed to recognize a person or multiple people in any picture, including photos and video frames. A *Convolutional Neural Network*, the powerful MobileNetV2, is used to extract features from images for the purpose.
Convolutional Neural Networks, or simply CNNs, are the most popular Deep Learning algorithms in the *Computer Vision* field, belonging to a more general context of *Machine Learning*.
This chapter offers a basic understanding of the interesting world of Machine Learning and, later, explores Deep Learning tecniques used in the development of the Deep One-class Classification algorithm, from Artificial Neural Networks to the amazing Convolutional Neural Networks.

2.1 Why Machine Learning

The first definition of Machine Learning is by Arthur Samuel [1] and dates back to 1959:

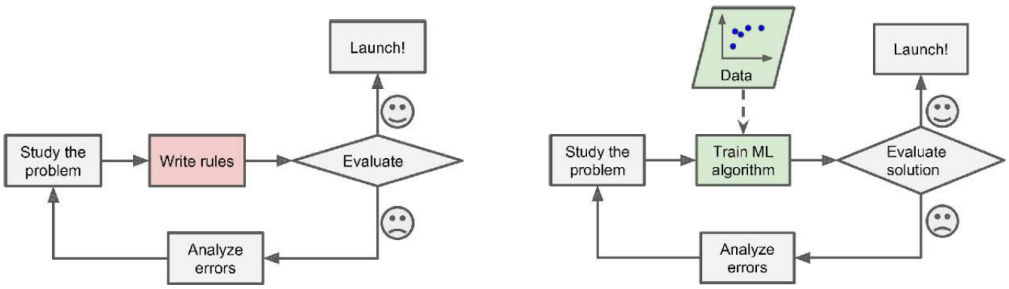
> *"Machine Learning is the field of study that gives computers the ability to learn without being explicitly programmed".*

A question naturally arises, how is it possible that a machine learns without guidelines defined by humans (*"being explicitly programmed"*)?
The answer exists and is to *rely on data*.
Actually, machine learning models are not built using pre-determined and stationary rules, but they evolve following available data structures. This is why Machine Learning approach differs from traditional ones.
Consider the following problem

$$y = f(x),$$

where x is the input, f is the function that represents the model and y is the output.
The *traditional approach* suggests us to write specific rules, through which we can identify a fixed function f. This function is fed with x and must generate an output close to y, for every couple of x and y. When the output is evaluated and it is far from the desired one, rules are re-written and the model is tested again.

The *Machine Learning approach*, instead, uses an iterative process of learning, relying on data. Model f is not locked and parameters are not strictly defined, continuosly changing to produce a predicted output that matches the desired one y.
In figure 2.1, the two approaches are sumarized.



(A) Traditional approach

(B) ML approach

FIGURE 2.1: Different approaches in solving a problem. *Source: A. Géron* [1]

It is necessary to cite an upcoming description of Machine Learning, provided by Tom Mitchell in 1997 [2]:

> *"A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E".*

The unique allowed strategy in this context is, therefore, to learn from experience E in order to improve performances related to a task T.

Machine Learning strategies are widely used nowadays and provide new services that were not feasible, even thinkable, before.
ML algorithms allow also to handle the enormous amount of data we are inexorably faced with today.
There are a lot of application fields of Machine Learning, from automotive to medicine, passing by health care, finance and space. Some practical examples can be medical image segmentation and classification to automate diagnosis process, self driving devices like Park Assist and Lane Keep Assist, speech recognition, person detection in video surveillance and so on.
Machine Learning permeates our lives and it is visibly changing the present. Machine Learning is definitely the future.

2.2 Some pieces of history

Some of the most significant steps that have brought Machine Learning where is today are here presented. In the figure below (fig. 2.2) a summarising timeline is shown:

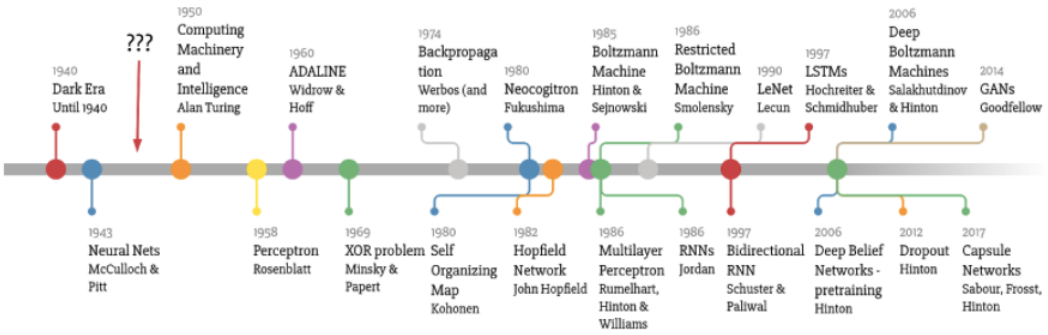


FIGURE 2.2: Machine Learning Timeline. *Source: F. Vázquez* [3]

Everything about Machine Learning starts in 1943, when a paper on human brain is published. In *"A Logical Calculus of Ideas Immanent in Nervous Activity"*, Warren McCulloch and Walter Pitts show how biological neurons might work. Later, they model a neural network with electrical circuits, which is the first artificial neuron emulating neuron activity in our brain.
The *Threshold Logic Unit* (TLU) is a later generalization of the McCulloch-Pitts model.

2.2.1 From 1950s to 1970s

About ten years later, in 1952, Arthur Samuel builds a computer program that learns from its mistakes and gets better while playing checkers.
In 1958, Frank Rosenblatt introduces the first pillar of current Neural Networks: the *perceptron*. It is a one-layer neural network which is able to recognize simple patterns and shapes associated to an image.
The following year, in 1959, in the wake of perceptron, Bernard Widrow and Marcian Hoff propose *ADALINE* - ADAptive LINear Element - at Stanford University. It is still a single layer neural network, where weights are tuned according to the weighted sum of the inputs.
Its evolution is called *MADALINE* - Many ADALINE - a three-layer fully connected artificial neural network that successfully manages to take off echoes over phone lines as adaptive filter. For the first time, a concrete problem of every day life is solved by an ANN.

In the next decade, we see a crucial decline in the interest of Machine Learning field and there are no steps forward in ML research.
The enthusiasm is curbed by the pubblication of the book *Perceptrons* written by Marvin Minsky and Seymour Papert, where some limitations of the Rosenblatt net are presented: it is capable of learning only linearly separable patterns, not even an XOR function.
Furthermore, the diffusion of Von Neumann architecture, easier to understand

than neural networks, and technological restrictions that slowed down the advance of multi-layer networks, bring to the standstill.

2.2.2 From 1980s to 2000s

Only in the 80s, research in neural network field takes hold again: the movement called *connectionism* makes his way, ending the "long winter".
In 1982, John Hopfield presents a network with bidirectional lines and in 1986 Geoffrey Hinton, in cooperation with David Rumelhart and Ronald J. Williams, discloses the revolutionary *backpropagation algorithm*.
Finally, multi-layer neural networks can be easly trained.
In 1997, the IBM computer *Deep Blue* beats the world chess grandmaster and in the next year, Yann LeCun at AT&T Bell Laboratories proposes a new type of neural networks, the actual *Convolutional Neural Networks*, that reaches good accuracy in digit recognition.
In 2006, the publication of the paper *"A Fast Learning Algorithm for Deep Belief Nets"* by Hinton, Simon Osindero and Yee-Whye Teh marks a resonable turning point and lays the foundation of a new field of Machine Learning, the Deep Learning. Training deeper and more complex multiple-layer networks is no longer infeasible.
Another important contibution in Deep Learning methods is the article *"Greedy Layer-Wise Training of Deep Networks"* by Yoshua Bengio et al. dated in 2007.
Geoffrey Hinton, Yoshua Bengio and Yann LeCun are actually referred to be the *"Godfathers of Deep Learning"*.

2.2.3 21st century

From then on, the power of Machine Learning was undeniable and gradually ML architectures overtook traditional methods in any field.
The very first important projects are, for example, *GoogleBrain* (2012) by Google, a deep neural network that is able to detect patterns in images and videos, *AlexNet* (2012) that introduces the use of GPUs and CNNs, and *DeepFace* (2014) by Facebook for person recognition.
All of that would not have been possible without:

- the increasing quantity of data available for training phase, that nowadays has reached million of examples in any application;
- the unthinkable growth of computational power, necessary to train deep Neural Networks;
- the improvement of training algorithms.

2.3 Artificial Neural Networks

Our brain allows us to constantly and efficiently learn. We are able to understand effects of our behavior and accordingly act, thanks to experience.

This is why researchers wanted to identify mechanisms behind human brains and, later, they tried to artificially reproduce them using simple mathematical models.
Artificial Neural Networks are designed just to emulate biological architecture of our brain. They belong to the category of *supervised learning algotithms*, where labeled data are present during the training: the desired output is available, in addition to his input, to properly guide the evolution of model parameters.

The basic unit of a neural network is the *neuron*. Inside an Artificial Neural Network there are a lot of neurons connecteted together: each of them takes as input some quantities and produces a single output that can be sent to other units.
Neurons are also organized in *layers*, creating a defined structure with the *input layer*, the first layer that acquires external data, the *output layer*, the last one of the network, and a variable number of *hidden layers* between them.
This kind of structure allows neurons from one layer, to have connections with only immediately previous and immediately next layers.
When all neurons of a layer have connections with all units of the following layer, the resulting Neural Network is said to be *fully-connected*.

Depending on the number of hidden layers, we can distinguish two types of nets (in figure 2.3): *shallow Neural Networks*, that have only one intermediate layer, and *deep Neural Networks*, that have a lot of hidden layers and are able to learn ever more complex functions.

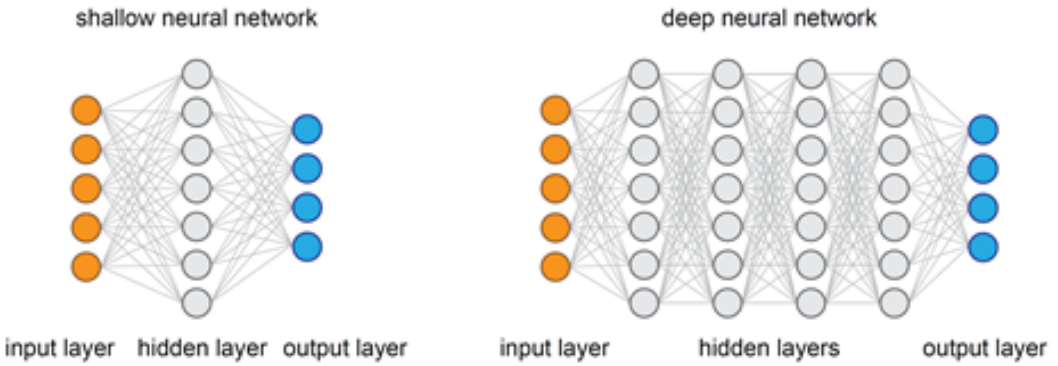


FIGURE 2.3: A shallow Neural Network and a deep Neural Network with 4 hidden layers. *Source: M. Terry-Jack* [4]

In order to understand the working mechanism of Artificial Neural Networks, the biological neuron stucture and the first artificial neuron models are presented.

2.3.1 Neuron models

A brain neuron is the main component of nervous tissue. His structure is shown is figure 2.4: it incudes the *soma*, that is the cell body, some *dendrites*, that are

filaments with tree shape, and the *axon*, a major extension. In the final part of the axon there are some *synapses*, little protuberances that make the axon in contact with other neurons, allowing the propagation of the electric impulse.

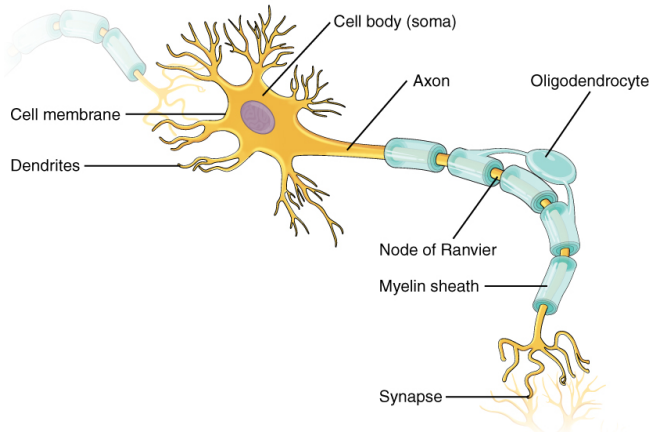


FIGURE 2.4: Neuron structure

Neurons receive electrochemical inputs at the dendrites and they send signals through the axon. These processes are both electrical and chemical.
It is foundamental to underline that a neuron is activated only if the sum of the electrical inputs is powerful enough to fire it. If the received signal does not surpass a certain threshold, the neuron does not transmit the output along the axon and the other neurons don't receive the above.

Making a weighted sum of inputs and subsequently determining a binary outcome if that sum overcomes a threshold or not, is an easy task executed by a single neuron. Our brain is made up of billions of neural cells which, performing this activity millions of times, carry out very complicated tasks.
This is the starting point to understand and shape an artificial neuron model.

McCulloch-Pitts neuron

It is presented in 1943 by Warren McCulloch and Walter Pitts and it is considered the first mathematical model of a neuron (fig. 2.5).
The structure has several binary inputs (one or more) and only one binary output. This kind of neuron is active when more than a fixed number of inputs are on. It follows that any logical preposition can be built with this model.

In the picture 2.5, x_i are the binary inputs, g is the sum of them, f is the decision function that acts in accordance with the number of active inputs (i.e. the threshold value) and y is the binary output.
For example, considering that a neuron fires when two inputs turn on, so the threshold is 2, simple boolean operations can be obtained, as shown in figure 2.6.

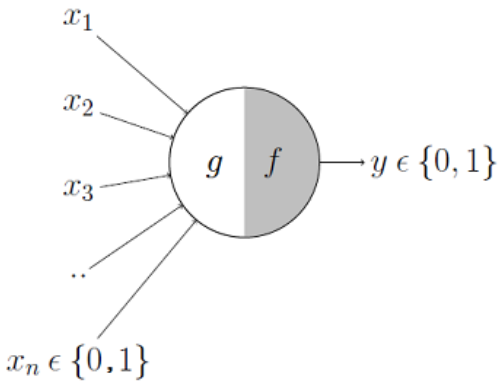


FIGURE 2.5: McCulloch-Pitts neuron. *Source: A. L. Chandra* [5]

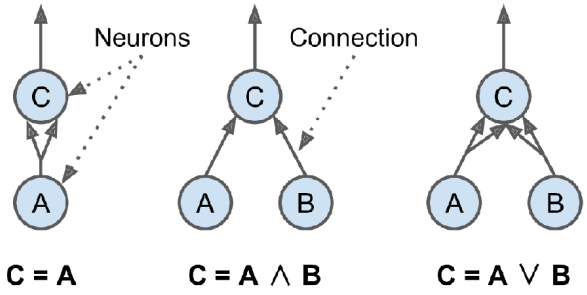


FIGURE 2.6: Simple boolean operations. *Source: A. Géron* [1]

Threshold Logic Unit

The *Threshold Logic Unit* (TLU), in figure 2.7, is a generalization of McCulloch-Pitts model. The key points of TLUs are the following:

- inputs are no longer binary on/off values but numbers;
- all inputs are modified by weights, that are all equal and determine the importance of each of them;
- inputs can be excitatory or inhibitory;
- a certain threshold θ and one binary output are still present;
- if there are no inhibitory signals, all weighted inputs are summed together and an activation function is applied: the output is 1 if the sum is greater than the threshold, otherwise it is 0.

The TLU behaviour is sumarized by this equation:

$$y = \begin{cases} 1 : \sum_{j=1}^{n} w_j x_j \geq \theta \ \wedge \ no\_inhibition \\ 0 : otherwise \end{cases}$$

Similarities between the biological neuron and TLU model are evident: synapses are represented by *links* among neurons and the electrical signal that flows

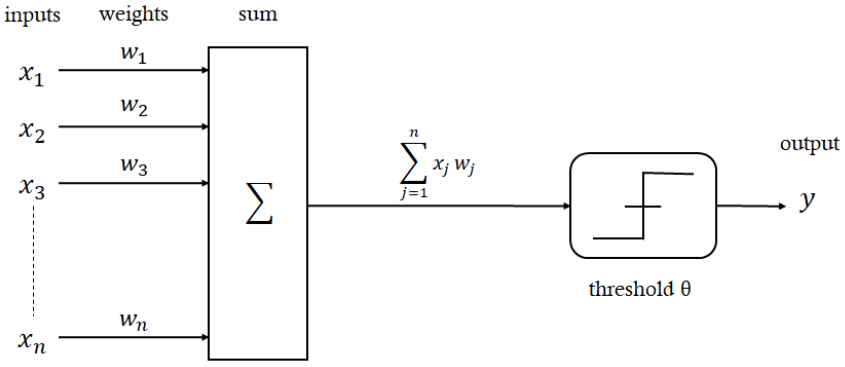


FIGURE 2.7: Threshold Logic Unit model

through axons is now a *number*. There are *weights* that determine the conductivity of the synapses and an *activation function* that plays the role of the threshold, beyond which the neuron fires.
What drives researchers to go further is the fact that this network does not learn. All weights and connections must be set manually, so there are no real benefits compared to standard methods.

Perceptron

The *perceptron* (fig. 2.8) is introduced by Frank Rosenblatt in 1958 and it is based on TLU with an extra input term fixed to 1.

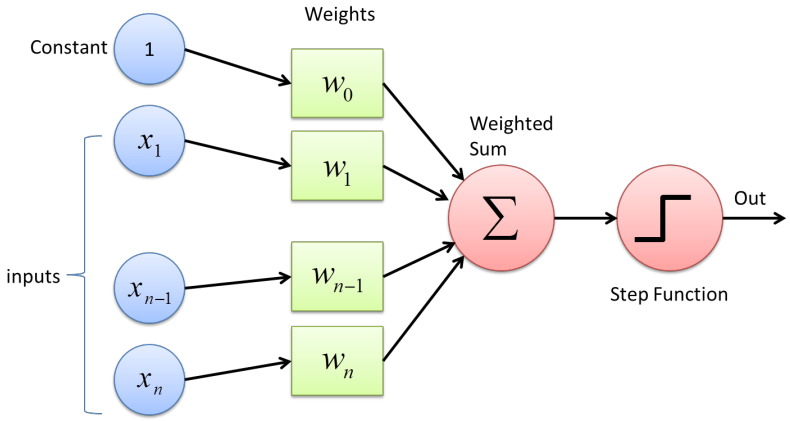


FIGURE 2.8: The perceptron. *Source: Deep AI* [6]

The key points of perceptrons are the following:

- the inhibitory synapses are gone;
- weight and threshold values can be different from each other;
- weights can have positive and negative values;
- finally a *learning algorithm* is present to automatically tune weights and biases.

The equation describing the perceptron behaviour is very close to the TLU one:

$$output = \begin{cases} 0 : \sum_{j=1}^{n} w_j x_j \leq threshold \\ 1 : \sum_{j=1}^{n} w_j x_j > threshold \end{cases}$$

In order to semplify the notation, we can make some modifications.
Considering w as the row vector of weights and x as the column vector of inputs, the summation can be replaced by a dot product: $w \cdot x \equiv \sum_j w_j x_j$.
Also, the threshold parameter is changed in perceptron's *bias*: $b \equiv$ *-threshold*.
The bias is something that indicates how difficult is to change the output from 0 to 1. Actually, when b is a very large positive number, it's very simple to have perceptron's output equal to 1; on the contrary, having a very negative b, makes it harder.
The equation is now:

$$\hat{y} = \begin{cases} 0 : w \cdot x + b \leq 0 \\ 1 : w \cdot x + b > 0 \end{cases}$$

The absolute innovation in Rosenblatt's perceptron, by returning to the last key point, is the introduction of a *learning algorithm*.
A network composed by perceptron units affords not only to compute any logical function, since it implements NAND gates that are universal for computation, but also to *learn* how to solve problems by its own!
It is no more necessary to set weights and biases by hands, they are tuned in an automatic way thanks to some learning rules.

The learning procedure takes inspiration from what Donald Hebb proposed in 1949 in his book *"The Organization of Behavior"*. The *Hebb's rule* is synthetized by Siegrid Löwel as *"Cells that fire together, wire together"*.
This means that when two neurons are activated together and several times, the connection between them is reinforced: this process is the foundation of learning.
It follows that neuron weights are updated looking at the error that the network does in making a prediction. Fixing a certain training instance, the *error* δ_j is the difference between y_j, the target output of the jth output neuron, and $\hat{y}_j$, the actual output of the jth output neuron.
The correction of the weight between the ith input neuron and the jth output neuron $\Delta w_{i,j}$ is affected by δ_j, that tells how far the actual output is from the desired output, by a quantity η called *learning rate* and by x_i, that is the ith input value of the current training instance.
Equations can be summarised as:

$$\begin{aligned} \delta_j &= y_j - \hat{y}_j \\ \Delta w_{i,j} &= \eta \cdot \delta_j \cdot x_i \\ w_{i,j}^{(next\_step)} &= w_{i,j} + \Delta w_{i,j} \end{aligned}$$

It is simple to see that connections which minimize δ_j are reiforced: if $\hat{y}_j$ is very far from y_j, the error is a large number and weights $w_{i,j}$ at the next step will be massively adjusted. Otherwise, if the difference is small, $\Delta w_{i,j}$ will be small too and weights are modified accordingly.

Sigmoid neurons

In a neural network, learning means that each unit changes its weight and bias in order to produce a wished final output.
If we want to reach good results, this procedure must be done gradually: weight values change bit by bit and the network behaviour gets little closer to the desired one, as shown in figure 2.9.

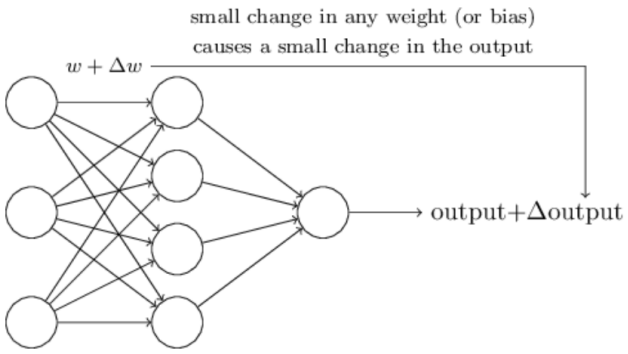


FIGURE 2.9: The learning procedure. *Source: M. Nielsen* [7]

These slow adjustments are not possible in a network of perceptrons!
The activation function of this kind of units is a *step function* (figure 2.10) and produces either 0 or 1, according to a certain threshold. Therefore the output is completely flipped, even with small changes in weights/biases. Sharp variations in network output causes blindness in learning steps.

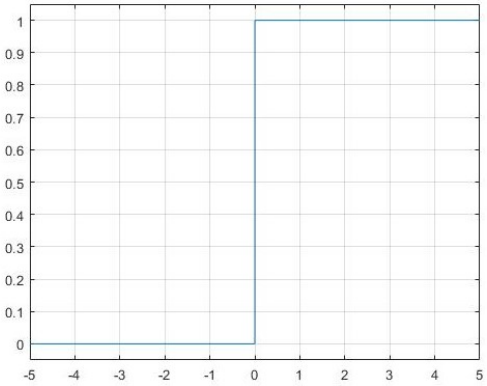


FIGURE 2.10: The Step function. *Source: A. Tartaglia* [8]

A new type of activation is needed in order to correctly learn: the *sigmoid function*, also called *logistic function*, represented in picture 2.11.

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

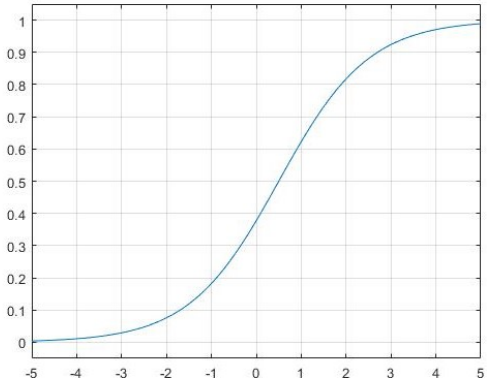


FIGURE 2.11: The Sigmoid function. *Source: A. Tartaglia* [8]

Neurons with the sigmoid activation are referred to be *sigmoid neurons*.
They have continuous outputs with any values between 0 and 1:

$$z = wx + b \longrightarrow \sigma(z) = \sigma(wx + b) = \frac{1}{1 + e^{-(\sum_i w_i x_i + b)}}$$

The sigmoid function is a soft version of the step activation and, most of all, it is differentiable in every point, essential requirement to compute the gradients and later tune the weights and biases.
A *sigmoid neuron* is used quite often as output unit of a Neural Network for binary classification problems.

Other activation functions

We have seen some activation functions like the *step function* in the perceptron and the *sigmoid function* in the sigmoid neuron.
Other activations are commonly used in Neural Networks and are presented hereafter.

Linear The *linear activation* is the simplest one: it creates an output identical to the input and it is represented by a straight line with slope equal to 1 (fig. 2.12).
It is like there is no activation at all, so:

$$linear(z) = a = z = \sum_{i=1}^{N} w_i x_i + b$$

Normally, linear activation functions are flanked by non-linear ones.
If we use only linear activation functions in the network, we will be able to compute solely linear functions, no matter how many layers are present or the whole structure complexity. A model with linear hidden layers implements just a standard logistic regression, like there were no hidden layers.

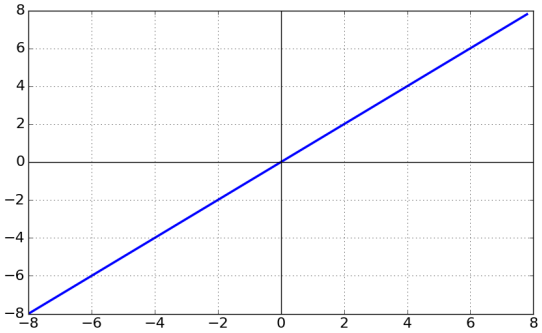


FIGURE 2.12: The linear function. *Source: S. Sharma* [9]

Elaborate mappings are learned by the model thanks to non-linearities and, therefore, the combination of various activation functions is aimed to introduce them in the Artificial Neural Network.

Tanh The *tanh function* is shown in figure 2.13 and its equation is:

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

It is very similar to the sigmoid function, but has a different output range: it is between -1 and 1 instead of 0 and 1.
The tanh function is widely used because it works better compared to sigmoid one: the mean of activations is close to 0, which signifies that data are zero-centered and makes the learning for next layers simpler.
Neurons with this activation function have this kind of output:

$$z = wx + b$$
$$a = \hat{y} = tanh(z)$$

The downside of both sigmoid and tanh functions is that small gradient is produced when z is very large or very tiny, corresponding to points with a flat slope. This slows down the *gradient descent* and also leads to the *vanishing gradient problem.*

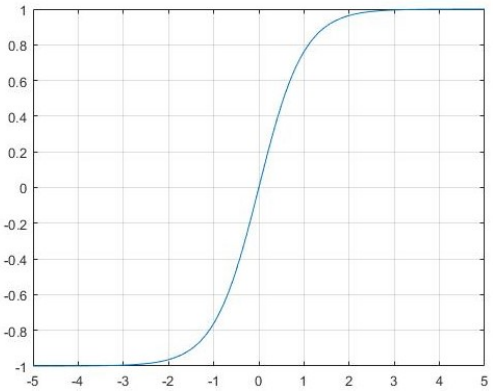


FIGURE 2.13: The tanh function. *Source: A. Tartaglia* [8]

ReLU *ReLU* stands for Rectified Linear Unit (fig. 2.14). Its equation is:

$$ReLU(z) = max(0, z)$$

When $z > 0$, the output is exactly z and the derivative is 1; when $z < 0$ the output is 0, as the unit is inactive, and the derivative is 0.
It is the default choice for the activation function in hidden units, because it produces better performance in training deep networks with respect to sigmoid and tanh activations. In practice, most of hidden units have $z > 0$, so the slope of ReLU is pretty different from 0, pushing the neural network to learn faster.

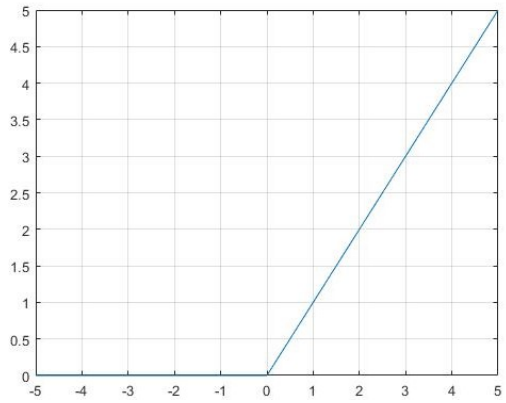


FIGURE 2.14: The ReLU function. *Source: A. Tartaglia* [8]

Leaky ReLU The *leaky ReLU activation* is an alternative version of ReLU. It is called "leaky" because it has a slight slope when z is less than or equal to 0. It is originated in order to overcome the non-differentiability problem of ReLUs, still giving a small quantity when $z < 0$.
The Leaky ReLU function is represented in figure 2.15 and its equation is:

$$leakyReLU(z) = max(\alpha z, z)$$

The parameter α is a small number, usually equal to 0.01.
This activation function works better than others, but is not utilized as much in practice.

Softmax The *softmax activation function* is generally used in the final layer of a multi-class classifier and can be seen as a generalization of the logistic regression.
This activation normalizes the output vector of a network in the way that all components belong to the intervall (0,1) and they sum up to 1. The obtained real numbers can be associated to the discrete probability distribution among classes, whose number matches the number of units in the output layer.

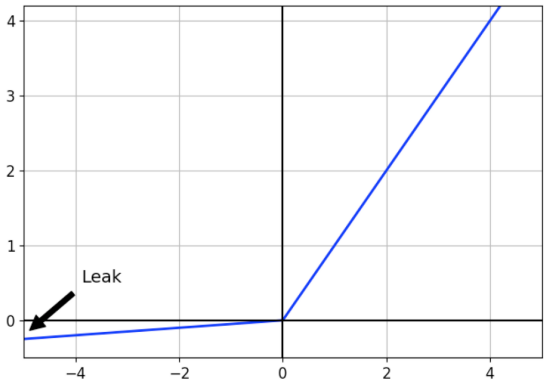


FIGURE 2.15: The leaky ReLU function. *Source:S. Sonawane* [10]

If we consider the last layer L that has N neurons, the softmax function returns:

$$softmax(z^L) = a^L = \frac{e^{z^L}}{\sum_{i=1}^{N} t_i},$$

where t_i are the elements of vector t, that is:

$$t = e^{z^L}.$$

Hence, each component of the activation a^L is:

$$a_i^L = \frac{t_i}{\sum_{i=1}^{N} t_i}$$

and represents the probability of an instance of belonging to a particular class since:

$$\sum_{i=1}^{N} a_i^L = 1.$$

Once explained the unit model and the different activations, it's time to combine all neurons to build a complete Artificial Neural Network (simply called Neural Network).

2.3.2 The architecture of Neural Networks

In the introduction part of section 2.3, some shallow and deep Neural Networks are shown.
They are both *feedforward Neural Networks*, because no loops are present in the model. Inside these nets the information moves only in one direction, from input layer, passing through hidden units, and never goes back. No cycles are allowed as well as there is no memory of signals: the output depends only on the actual input.

On the opposite side, there are *recurrent Neural Networks*. Here loops are admitted and output values of an higher level layer can be used as input of an erlier layer. Therefore, a temporal dynamic behaviour is established.
In this work of thesis only feedforward Neural Networks will be treated, by referring to them as Neural Networks.

First of all, a clarification on Neural Network *parameters* is presented.
We have already seen that each neuron performs two actions with its inputs: it makes a weighted sum, adding a bias terms, and applies to the latter an *activation function* σ:

$$z = w \cdot x + b$$
$$y = a = \sigma(z)$$

Recall that only the first equation contains vectorial quantities, that are x, the column vector of inputs, and w, the row vector of weights.
Other terms are, instead, real numbers: b is the bias term, z is the intermediate output to which function σ is applied and a or y is the output of the neuron.

A Neural Network is composed by several neurons, organized in layers.
From now on, all single parameters will have an *apex*, to denote the layer the parameter belongs to, and a *subscript*, to identify which neuron the parameter is referred to.
Additionally, L is commonly used to specify the total number of layers and n^l is the number of units in layer l.
The equations that describe the behaviour of any layer l in an Artificial Neural Network are:

$$z^l = W^l a^{l-1} + b^l$$
$$a^l = \sigma(z^l).$$

In particular parameters of a generic layer l are:

- $a^{l-1} \in (n^{l-1}, 1)$: vector of activations of previous layer, that is the input vector of the lth layer;
- $W^l \in (n^l, n^{l-1})$: matrix of weights of current layer l that links a^l and a^{l-1};
- $b^l \in (n^{l-1}, 1)$: vector of biases of current layer l;
- $a^l \in (n^{l-1}, 1)$: vector of activations of current layer l.

In figure 2.16 a Neural Network with some highlighted parameters is shown: it has three neurons in the input layer (n^0=3), five neurons in the 1st and 2nd hidden layers (n^1=n^2=5), three neurons in the 3rd one (n^3=3) and the last layer has one output neuron (n^4=1). Notice that activations of layer 0 corresponds to the input vector x, $a^0 \equiv x$.

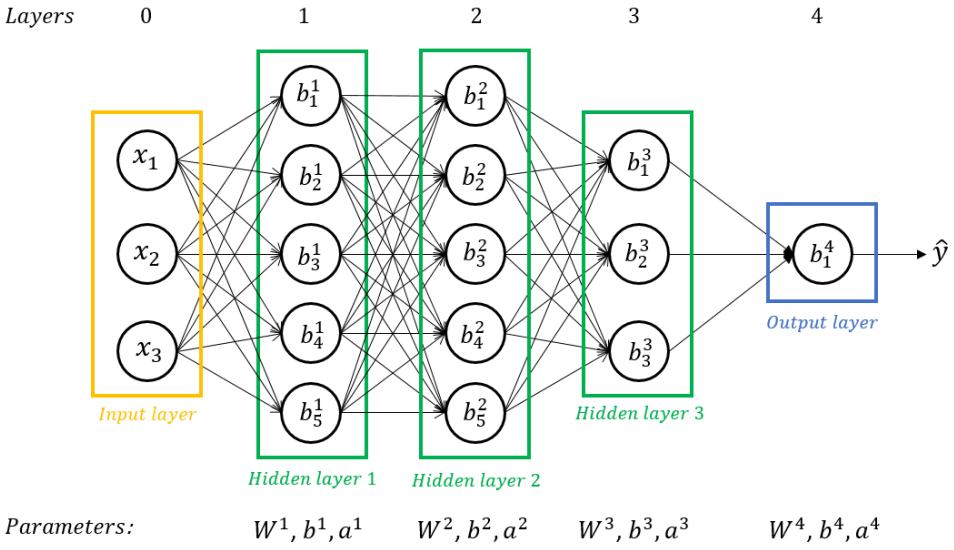


FIGURE 2.16: Parameters of a Neural Network

As an example, equations of the first two hidden layers and of the output layer are shown:

$$z^1 = W^1 a^0 + b^1$$
$$a^1 = \sigma(z^1)$$

$$z^2 = W^2 a^1 + b^2$$
$$a^2 = \sigma(z^2)$$

and

$$z^4 = W^4 a^3 + b^4$$
$$a^4 = \sigma(z^4).$$

The size of each parameter is:

$$\begin{aligned} a^0 &\in (3,1) \\ z^1, b^1, a^1 &\in (5,1) \\ W^1 &\in (5,3) \\ \\ z^2, b^2, a^2 &\in (5,1) \\ W^2 &\in (5,5) \\ \\ a^3 &\in (3,1) \\ z^4, b^4, a^4 &\in (1,1) \\ W^4 &\in (1,3). \end{aligned}$$

2.3.3 Learning in Artificial Neural Networks

So far, all models and parameters characterizing an Artificial Neural Network have been exposed.
lt's time now to describe the main ingredients that bring Neural Networks to correctly tune these parameters and, actually, to *learn*.

Cost function

The first essential component to train weights and biases is the *cost function*, also called *loss function*.
The cost function is a quantity that is periodically evaluated during trainings because it quantifies how well the learning is going.
It is computed using the *actual predictions*, coming as outputs from the network, and the *training examples*, m input-output pairs that belong to the *training set*:

$$\{(x_1, y_1), (x_2, y_2), ..., (x_m, y_m)\}.$$

Using the subscript i to consider the ith example in the training set, generic quantities are:

- x_i: actual input of the network;
- y_i: desired and known output related to x_i;
- $\hat{y}_i$: actual predicted output when x_i is fed, i.e. $\hat{y}_i$=σ(wx_i+b).

During the training, the generic input x_i passes through the entire Neural Network and produces the output $\hat{y}_i$.
At the end of this *forward propagation step*, the cost function *C(w,b)* is computed as the *error* between the target output y_i and the predicted one $\hat{y}_i$.
This quantity depends on the collection of all weights in the network and on all the biases and it must be minimized, because we want $\hat{y}_i \approx \hat{y}_i$.
The procedure ends when parameters w and b that minimize the overall cost

function are found.
Learning, in summary, is to adjust all weights and biases until the network produces the desired outputs.

A classical cost function is the *quadratic cost function*, known also as *Mean Square Error* and *L2 loss*:

$$C(w,b) = MSE(w,b) = \frac{1}{2m}\sum_{i=1}^{m}(\hat{y}_i - y_i)^2.$$

It is the average of squared differences between predictions and actual outputs. It is always positive because each term of the sum is positive. Consequently, minimizing *C(w,b)* is equal to have *C(w,b)* ≈ 0, where each squared difference tends to 0, since $\hat{y}_i \approx \hat{y}_i$ for all m training inputs.
The goal of this training is to find parameters w and b for which *C(w,b)* ≈ 0.

Other cost functions

In addition to the Mean Square Error, there are lots of commonly used loss functions. No absolute rules exist to determine the loss function cut out for us. Some of crucial factors can be: the type of problem to solve, a *regression* or a *classification* probem, the configuration of the output layer that leads to a *binary* or to a *multi-class* classification problem, the accuracy to reach, and so on.

Mean Absolute Error The *Mean Absolute Error* is the average of the sum of absolute differences between predictions and actual outputs:

$$MAE(\hat{y}, y) = \frac{1}{m}\sum_{i=1}^{m}|\hat{y}_i - y_i|.$$

It is also called *L1 loss* since the magnitude, instead of square, is computed. The MAE cost is more robust against outliers than MSE one, but its gradient is not easy to calculate.

Cross Entropy The *cross entropy loss* is used in classification problems, in which outputs are probability distributions between 0 and 1.
The loss is defined as

$$CE(\hat{y}, y) = -\sum_{i=1}^{C} y_i log(\hat{y}_i), \tag{2.1}$$

where y_i is the ground truth vector, $\hat{y}_i$ is the estimated output coming from the network and C is the number of different classes in our problem.
When the number of classes in equation 2.1 is equal to 2, we are faced with a *binary classification* problem and the loss is:

$$CE(\hat{y}, y) = -\sum_{i=1}^{C=2} y_i log(\hat{y}_i) = -y_1 log(\hat{y}_1) - y_2 log(\hat{y}_2).$$

It can be re-written simply as

$$CE(\hat{y}, y) = -ylog(\hat{y}) - (1-y)log(1-\hat{y}),$$

because the output neuron has a sigmoid activation, that gives only two probabilities with a sum equal to 1: $y_1 = 1 - y_2$ and $\hat{y}_1 = 1 - \hat{y}_2$.
In this way,

- if $y = 1$: the loss becomes $-log(\hat{y})$ and its minimization implies a large value of $\hat{y}$, so $\hat{y}$ close to 1;
- if $y = 0$: the loss is $-log(1-\hat{y})$ and its minimization implies a small value of $\hat{y}$, so $\hat{y}$ close to 0.

Optimization algorithm: Gradient Descent

The process of minimization can be easly visualized when parameters are less than two.

In the image below (fig. 2.17), for example, the cost function C has only one dimension, referred as v_1. The minimum of $C(v_1)$ is the red dot and corresponds to v_1^*.

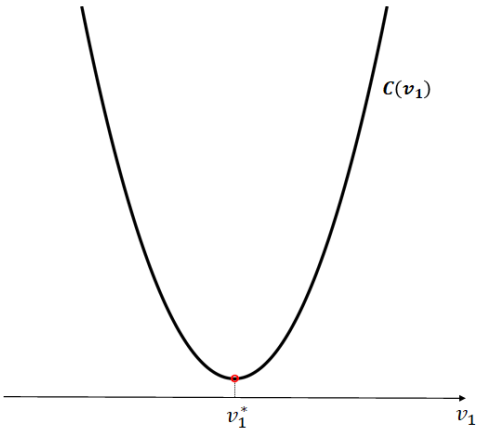


FIGURE 2.17: Minimization of a cost function with one parameter

In picture 2.18, instead, C depends on two parameters, v_1 and v_2. It can be depicted as a surface plotted in the plane generated by (v_1, v_2) and its minimum is again highlighted in red.

Generally, minimizing a cost function $C(w,b)$ in Artificial Neural Networks is not a simple procedure because the number of dimensions is higher than two. There are plenty of parameters which are all weights and all biases introduced by neurons.
Since calculus in minimizing this type of cost functions is not effective, another approach is taken: updating parameters to reach the minimum using an *optimization algorithm.*

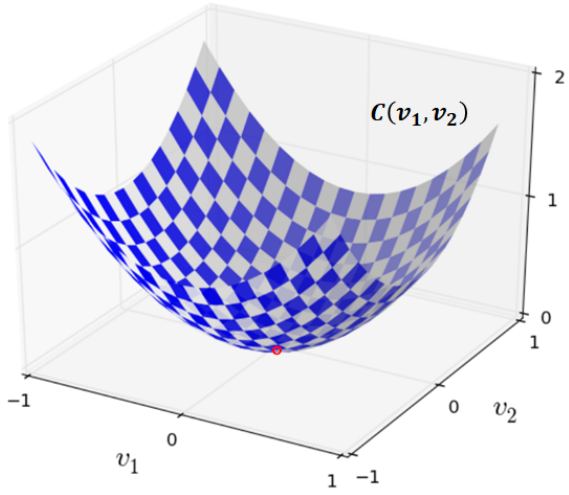


FIGURE 2.18: Minimization of a cost function with two parameters. *Source: M. Nielsen* [7]

We will firstly analyze the easiest algorithm, called *gradient descent*.
This optimizer goes "downhill" on a cost function C thanks to the gradient computation.
At first, the mechanism is described qualitatively using the one-dimensional case in picture 2.19. The presented cost function *C(w)* has one parameter for the sake of simplicity.

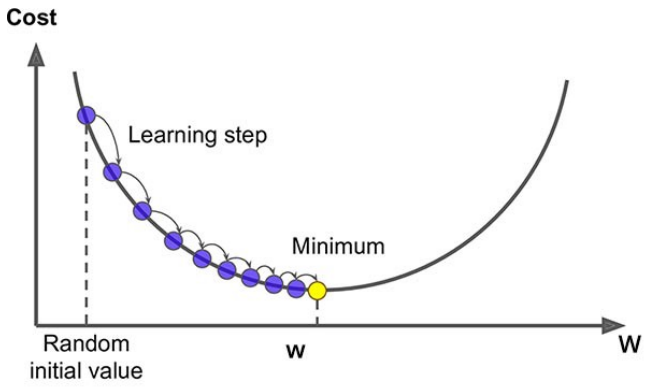


FIGURE 2.19: Gradient descent in unidimensional case. *Source: S. Bhattarai* [11]

We start from a random point in the left part of the curve. The derivative of the cost function with respect to the parameter w is computed as

$$\frac{dC(w)}{dw}$$

and represents the slope of the function in that point, which is negative. In order to move towards the minimum direction, we have to go to the right, making a

positive update with a bigger w. The operation done by gradient descent is:

$$w := w - \eta \frac{dC(w)}{dw}.$$

The parameter η is the *learning rate* and controls how big is the step we make at each iteration. It is always a positive real number.
This update makes sense even when the random point belongs to the right part of the curve. The derivative is now positive and, according to the formula, the parameter at next iteration becomes smaller. We move therefore to the correct left part, following the minimum direction.

A more rigorous discussion is presented below, generalizing the gradient descent mechanism to several parameters.
A cost function $C(\boldsymbol{v})$ that depends on m variables $\boldsymbol{v}=(v_1,v_2,\ldots, v_m)$ is considered. If each parameter v_i has a variation of Δv_i, also the the cost function C has a change, quantified as:

$$\Delta C \approx \frac{\partial C}{\partial v_1}\Delta v_1 + \frac{\partial C}{\partial v_2}\Delta v_2 + \cdots + \frac{\partial C}{\partial v_m}\Delta v_m. \tag{2.2}$$

The previous equation is made compact, defining the array of all variations $\Delta \boldsymbol{v}$ and the gradient of the cost function ∇C as the vector of the partial derivatives, as follows:

$$\Delta \boldsymbol{v} \equiv (\Delta v_1, \Delta v_2, ..., \Delta v_m)^T$$
$$\nabla C \equiv (\frac{\partial C}{\partial v_1}, \frac{\partial C}{\partial v_2}, ..., \frac{\partial C}{\partial v_m})^T.$$

The equation 2.2, therefore, becomes:

$$\Delta C \approx \nabla C \cdot \Delta \boldsymbol{v}.$$

Remember that the aim of the gradient descent is to minimize the cost function acting on the parameters: it is finally time to choose $\Delta \boldsymbol{v}$, the variation of parameters, in the way that it makes ΔC, the variation of the cost function, negative.
Supposing to have

$$\Delta \boldsymbol{v} = -\eta \nabla C,$$

the preceding equation is re-written in this way:

$$\Delta C \approx -\eta \nabla C \cdot \nabla C = -\eta ||\nabla C||^2.$$

If we change $\boldsymbol{v}$ like above, C will always decrease and never increase: ΔC is forced to be negative because the quantity $||\nabla C||^2$ is always greater or equal to 0. The goal is reached.

The update rule coincides with the one submitted in the semplified case and can be sumarized for all weights and biases as:

$$w_i := w_i - \eta \frac{\partial C}{\partial w_i}$$
$$b_i := b_i - \eta \frac{\partial C}{\partial b_i}.$$

Updating parameters in this way, in the opposite direction of the gradient, decrease C up to the global minimum.

Particular attention must be paid to the choice of *learning rate*: a small value of η can require too much time for training, while a large value of η makes computations fast, but can lead to an unstable behaviour with $\Delta C > 0$.

Other optimization algorithms

Using an optimization algorithm enables to easily train our Artificial Neural Networks. Large dataset with several training examples can make this procedure slow, so selecting a good optimizer is foundamental to speed up the entire training.
Variants of the gradient descent and other optimizers are here presented.

Batch gradient descent The tecnique called *vectorization* lets us to manage multiple elements from the training set. Each of them is a (x,y) pair forming X, that is the collection of input vectors of the network and Y, that is the collection of the related known target outputs. Consequently, our set is:

$$X \in (n_x, m) = [x^1, x^2, x^3, x^4, ..., x^m]$$
$$Y \in (1, m) = [y^1, y^2, y^3, y^4, ..., y^m].$$

If the gradient descent is applied to the whole training set, we call it *batch gradient descent*. In this scenario, one step of gradient descent is taken only after all m examples are processed. The entire procedure is very slow for large dataset, even when using the vectorization, and it is not feasible if the data don't fit in memory.
The batch GD proceeds to the minimum with big steps without oscillations, like shown in figure 2.21(A).

Mini-batch gradient descent In order to overcome issues related to batch gradient descent, the dataset is splitted into smaller training sets, known as *mini-batches* with a size, obviously, lower than the one of the entire dataset (usually from 64 to 512).
For example, considering the size of each mini-batch equal to 64, the set is splitted as shown in figure 2.20.
In this scenario we can use the *mini-batch gradient descent*, where one mini-batch at time is processed.
Since a mini-batch contains a defined number of training examples, they are all

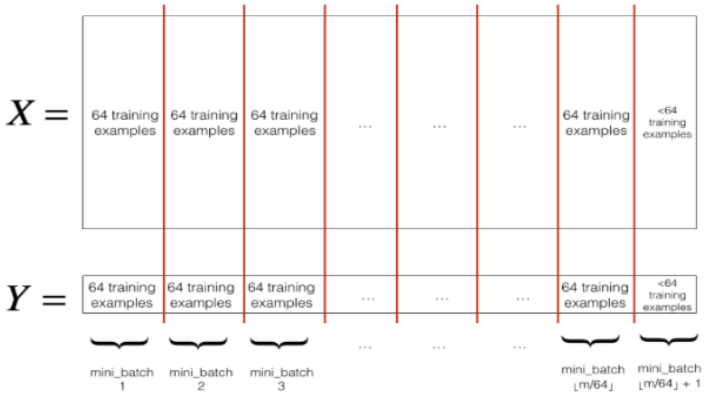
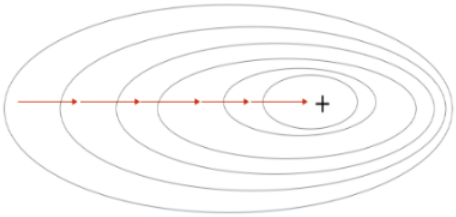


FIGURE 2.20: Training set splitted in mini-batches. *Source: Andrew Ng* [12]

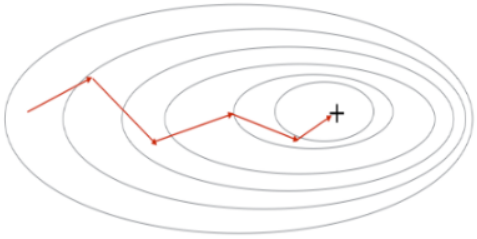
fed in parallel in the network and the progress in updating parameters is done after each mini-batch is processed.
In figure 2.21(B), we see progressive updates with some oscillations typical of mini-batch gradient descent: the cost function might not decrease at every iteration because a particular mini-batch could have an higher cost with respect to the previous one.
Anyway, it is necessary to define the quantity called *epoch*, that indicates a single pass through the training set. Our training will last many epochs till the convergence of the algorithm.
The mini-batch gradient descent is the most used method for training deep Neural Networks.

Stochastic gradient descent If the mini-batch size is 1, we are facing the *stocastic gradient descent*. Here, parameter update is performed after each training example is processed, so the speed up of vectorization is lost.
Frequent updates with high variance lead to high oscillations, like shown in figure 2.21(C), and sometimes the global minimun is never reached because SGD can fall in local minima.

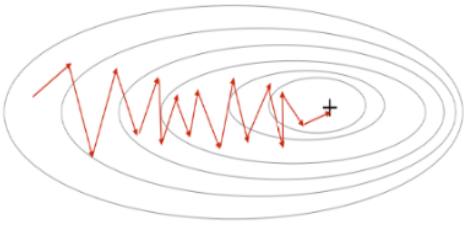
Momentum This method works faster than standard gradient descent bacause it keeps track of the direction of gradients. The *momentum* allows a faster learning along the direction of the minimum and damps oscillations along the other one, that usually slow the process.
The quantity that stores past updates is the exponentially weighted average of the gradient at previous steps, represented in the equations by V_{dw} and V_{db}.
At each iteration, $dw = \dfrac{\partial C}{\partial w}$ and $db = \dfrac{\partial C}{\partial b}$ are computed on current mini-batch



(A) Batch gradient descent



(B) Mini-batch gradient descent



(C) Stochastic gradient descent

FIGURE 2.21: Variants of gradient descent. *Source: Andrew Ng* [12]

and, then, the updates are performed:

$$
\begin{aligned}
V_{dw} &= \beta V_{dw} + (1-\beta)dw \\
V_{db} &= \beta V_{db} + (1-\beta)db \\
w &:= w - \eta V_{dw} \\
b &:= b - \eta V_{db}.
\end{aligned}
$$

The momentum is characterized by the parameter β, that controls the exponentially weighted average. For example, the last 10 gradients will be considered in the average if $\beta = 0.9$.
The term η is, instead, the well know learning rate.

RMSprop Also the *Root Mean Square prop* method speeds up the gradient descent.
Here, the exponentially weighted average of the squares of derivatives is stored in S_{dw} and S_{db} and updates contain these terms to adapt the learning rate. At each iteration, $dw = \dfrac{\partial C}{\partial w}$ and $db = \dfrac{\partial C}{\partial b}$ are computed on current mini-batch

and, then, the updates are performed:

$$S_{dw} = \beta_2 S_{dw} + (1-\beta_2)dw^2$$
$$S_{db} = \beta_2 S_{db} + (1-\beta_2)db^2$$
$$w := w - \eta \frac{dw}{\sqrt{S_{dw}+\epsilon}}$$
$$b := b - \eta \frac{db}{\sqrt{S_{db}+\epsilon}}.$$

In this way, η can be a large number without diverging, while ϵ is introduced to ensure numerical stability.

Adam The *Adam optimizer* is the most effective in training Neural Networks with very different architectures. It puts together benefits from RMSprop and momentum.
It computes the exponentially weighted average of past gradients through V_{dw} and V_{db} and the exponentially weighted average of the squares of past gradients through S_{dw} and S_{db}. A correction is made on all parameters and, finally, the update is done as follows:

$$V_{dw} = \beta_1 V_{dw} + (1-\beta_1)dw, \qquad V_{db} = \beta V_{db} + (1-\beta)db$$
$$S_{dw} = \beta_2 S_{dw} + (1-\beta_2)dw^2, \qquad S_{db} = \beta_2 S_{db} + (1-\beta_2)db^2$$
$$V_{dw}^{corrected} = V_{dw}/(1-\beta_1), \qquad V_{db}^{corrected} = V_{db}/(1-\beta_1)$$
$$S_{dw}^{corrected} = S_{dw}/(1-\beta_2), \qquad S_{db}^{corrected} = S_{db}/(1-\beta_1)$$
$$w := w - \eta \frac{V_{dw}^{corrected}}{\sqrt{S_{dw}^{corrected}+\epsilon}}, \qquad b := b - \eta \frac{V_{db}^{corrected}}{\sqrt{S_{db}^{corrected}+\epsilon}}.$$

Backpropagation

We have seen that the *gradient descent* is able to minimize a cost function C, updating parameters with gradient computation.
The gradient contains partial derivatives of the cost function with respect to each of all parameters: $\frac{\partial C}{\partial w_{jk}}$ and $\frac{\partial C}{\partial b_j}$.
In Artificial Neural Networks the number of variables is very huge, because a single neuron introduces a weight w_{jk} and a bias b_j and there are many layers with a lot of units.
Consequently, the computation of every single derivative is unfeasible.
The *backpropagation* algorithm allows to calculate the gradient one layer at time in an efficent way, reusing quantities already computed and iterating backward from the last layer through the chain rule.

The four fondamental equations of the backpropagation algorithm are the following:

$$
\begin{aligned}
&1. \quad \delta^L = \nabla_a C \odot \sigma'(z^L) \\
&2. \quad \delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l) \\
&3. \quad \frac{\partial C}{\partial b^l_j} = \delta^l_j \\
&4. \quad \frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j
\end{aligned}
$$

First of all, the term δ^l is the error vector associated with the lth layer, composed by all errors δ^l_j of the jth neurons in that layer.
Starting from the output error δ^L, subsequent quantities δ^l are derived and, then, partial derivatives are computed using the third and fourth equations.

In particular,

- the 1*st equation* computes the output error δ^L and is presented in a matrix-based form. The symbol $\odot$ indicates the *Hadamard product*, that is the elementwise product of two vectors.
  Each element of δ^L is

  $$\delta^L_j = \frac{\partial C}{\partial a^L_j} \sigma'(z^L_j),$$

  where $\frac{\partial C}{\partial a^L_j}$ indicates how fast the cost function changes with respect to the jth output activation a^L_j and $\sigma'(z^L_j)$ measures how fast the activation function σ changes with respect to z^L_j.
  Therefore, $\nabla_a C$ is a vector that incorporates all $\frac{\partial C}{\partial a^L_j}$.
  Elements of the above equation can be easily computed and some of them depend on the form of the chosen cost function.

- The 2*nd equation* permits the computation of the error δ^l for any layer l in the network. Each one is obtained through the known quantity δ^{l+1} at next layer $l+1$, carrying the error backward across all layers.

- The 3*rd equation*, instead, describes how the cost changes with respect to any bias. The derivative is computed thanks to δ^l_j given by the previous equation.

- In the last equation, the 4*th equation*, the rate of the change of C with respect to any weight is shown. The effect is that, when the activation of the neuron input is small, the gradient is small too and the weight has no big changes during the gradient descent.

Backpropagation: a practical sample

An implementation of the backpropagation algorithm is now presented through a practical sample.

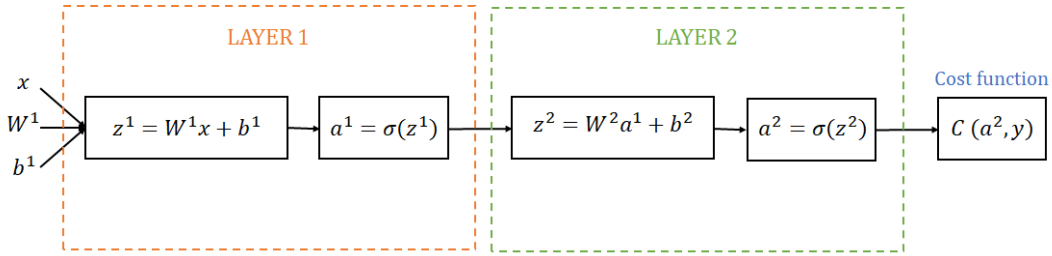


FIGURE 2.22: A two-layer Neural Network

The above Artificial Neural Network has only two layers (L=2) for simplicity, whose quantities are marked by an apex.
The steps are:

- *Input*: the input x is fed in the network.
- *Feedforward*: for each layer, z^l and a^l are computed:

$$z^1 = W^1x + b^1 \tag{2.3}$$
$$a^1 = \sigma(z^1) \tag{2.4}$$
$$\tag{2.5}$$
$$z^2 = W^2a^1 + b^2 \tag{2.6}$$
$$a^2 = \sigma(z^2). \tag{2.7}$$

  At the end, the cost function $C(a^2,y)$ is calculated.

- *Outputterror*: δ^L is computed:

$$\delta^2 = \frac{\partial C}{\partial z^2} = \frac{\partial C}{\partial a^2}\frac{\partial a^2}{\partial z^2} = \frac{\partial C}{\partial a^2}\sigma'(z^2)$$

- *Backpropagation*: for each layer δ^l is computed:

$$\delta^1 = \frac{\partial C}{\partial z^1} = \frac{\partial C}{\partial a^1}\frac{\partial a^1}{\partial z^1} = \frac{\partial C}{\partial a^1}\sigma'(z^1).$$

  The quantity $\dfrac{\partial C}{\partial a^1}$ can be re-written as:

$$\frac{\partial C}{\partial a^1} = \frac{\partial C}{\partial z^2}\frac{\partial z^2}{\partial a^1} = \delta^2 W^2$$

  and so,

$$\delta^1 = W^2\delta^2\sigma'(z^1).$$

Here, the already computed error δ^2 is reused at earlier layer in the computation of error δ^1.

- *Output*: gradients are given by

$$\frac{\partial C}{\partial W^2} = \frac{\partial C}{\partial z^2}\frac{\partial z^2}{\partial W^2} = \delta^2 a^1$$
$$\frac{\partial C}{\partial b^2} = \frac{\partial C}{\partial z^2}\frac{\partial z^2}{\partial b^2} = \delta^2$$

because, from equation 2.6, we have that $\frac{\partial z^2}{\partial W^2} = a^1$ and $\frac{\partial z^2}{\partial b^2} = 1$.

2.4 Convolutional Neural Networks

Convolutional Neural Networks are a type of deep Neural Networks aimed at dealing with images.
It is not easy to manipulate images with Artificial Neural Networks: there are too many parameters to handle.
In order to explain why, the representation of a digital image is introduced: it is a matrix of pixels, where each pixel contains information about the color intensity in that point.
In *grayscale* images the pixel is characterized by only one value, while in *color* (or *RGB*) images the pixel has three values corresponding to red, green and blue channels.
If an RGB image with 1000×1000 pixels is directly fed into a Neural Network, the input vector X has a dimension equal to $1000 \times 1000 \times 3 = 3millions$.
Considering also a rich hidden layer with 1000 units, weights are in the matrix $W^1 \in (1000, 3m)$, so $3billions$ of parameters are present just to start.
Here, Convolutional Neural Networks come to the aid and help us to face this parameter turmoil.
Another big difference between ANNs and CNNs is how images are acquired.
Networks with only fully connected layers take pixels, each one above the other, flattening the picture and without giving a real meaning to it.
On the contrary, the strength of Convolutional Neural Networks is to take into consideration the *spatial structure* of the images. That's why CNNs are so high-performing in image classification and recognition.
In the following sections, the fundamental concepts of Convolutional Neural Networks are investigated.

2.4.1 Convolutional layers

Convolutional Neural Networks are still inspired to biological processes, in particular to mechanisms of cells in the visual cortex. These neurons fire when specific geometric shapes, like horizontal lines or stripes with particular angles, are placed in the visual field.
Layers in CNNs behaves on the same page: early ones detect simple lines like edges, some later ones detect parts of objects and even later layers detect parts

of complete objects. An example is shown in figure 2.23: starting from edges, parts of faces (eye, nose, ear, ...) and cars (tyre, window, taillight, ...) are detected, till entire objects in last layer.

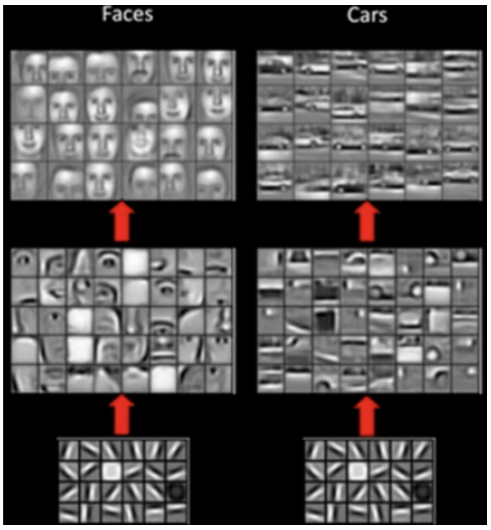
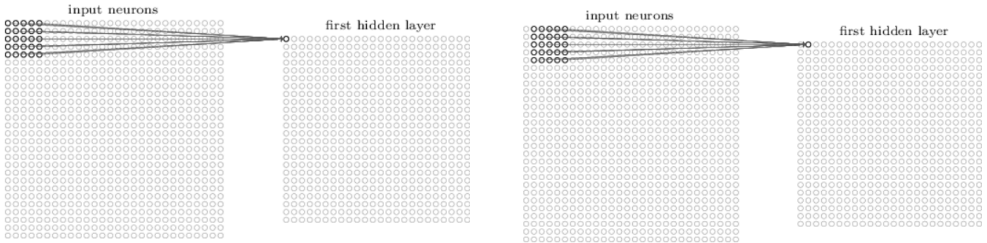


FIGURE 2.23: Features extracted through CNN layers. *Source: M. Stewart* [13]

It raises a question: how the simplest operation of detecting edges in images is performed? The answer is using an operation called *convolution* that is applied in convolutional layers of the network.

2D convolutions

At the beginning, we perform (for semplicity) convolutions on grayscale images, known as *2D convolutions*.
Since in CNNs the spatial structure of images is kept, the inputs is now a matrix of pixels, that can be figured as a square of neurons.
Unlike the Artificial Neural Networks where every input pixel is connected with every hidden neuron, in CNNs each neuron of the hidden layer is connected to only a portion of the area of input neurons, named *local receptive field*. It can be seen as a square window that slides through the input image.
In the picture 2.24 the first two steps of the procedure are shown: the 5×5 win-



(A) First step (B) Second step

FIGURE 2.24: First two steps in CNNs using 5×5 receptive field. *Source: M. Nielsen* [7]

dow starts in the top-left corner and moves across the space of 28×28 pixels, varying the hidden neuron connected to the square portion. The resulting layer has 24×24 neurons because the receptive field is moved by one pixel at time. This parameter is known as *stride* and determines the length of the taken step, so can differ from 1.
Every single hidden unit takes information from its receptive field, learning a weight for each connection and a unique bias. In particular, the operation executed by each neuron is a weighted sum of the inputs, computed by the *convolutional operator*, followed by a bias addition and the application of an activation function.
The number of learned weights corrisponds exactly to the number of pixels in the receptive field. Also, the weights and the bias are the same for all the hidden neurons of a layer, peculiarity inside all hidden layers of CNNs known as *shared weights and biases*.
These quantities are not fixed and are learned as parameters in order to detect a specific feature, like edges, in different positions of the input image.
The map that connects the input layer to the hidden layer is actually called *feature map*, but also terms like *kernel* and *filter* are used to refer to it.
A convolutional layer can be composed by multiple different feature maps that detect various features in the image.
The output of a convolutional layer is formed by several matrices where these features are highlighted, which are still images.

In the following sample, we dwell on convolutional operation in vertical edge detection.
As shown in figure 2.25, the input is a m×m=6×6 grayscale image, the feature map is the f×f=3×3 matrix, while the resulting image is a matrix of m-f+1×m-f+1= 4×4 pixels. The images corresponding to these matrices of pixel intensities are displayed down.

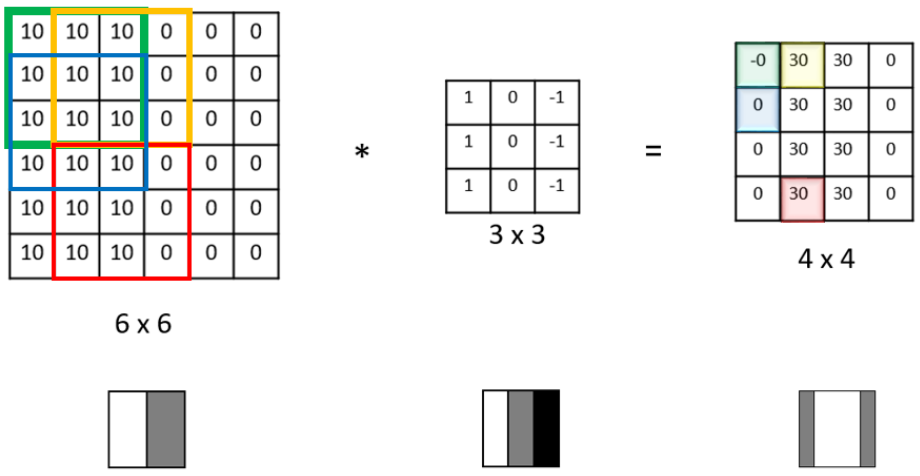


FIGURE 2.25: Convolutions in vertical edge detection. *Source: Andrew Ng* [12]

The feature map is firsty applied on the top-left corner in the green position. The convolutional operation consists in multipling each number of the filter with the corresponding one in the input matrix and summing all products, this is the weighted sum of inputs performed by first hidden neuron. The resulting number is put in the output matrix in first green square:

$$\begin{aligned} &10 \times 1 + 10 \times 0 + 10 \times -1 + \\ &10 \times 1 + 10 \times 0 + 10 \times -1 + \\ &10 \times 1 + 10 \times 0 + 10 \times -1 = 0. \end{aligned}$$

The filter is then shifted to the right in yellow position, where another convolution is performed:

$$\begin{aligned} &10 \times 1 + 10 \times 0 + 0 \times -1 + \\ &10 \times 1 + 10 \times 0 + 0 \times -1 + \\ &10 \times 1 + 10 \times 0 + 0 \times -1 = 30. \end{aligned}$$

The result is the yellow value of the second hidden neuron.
The window is iteratively moved with a stride length of 1, up to fill the output matrix, passing through blue and red positions.
The presented feature map detects vertical edges, because it creates in the output image a lighter region in correspondence of the vertical transition from white pixels (marked by 10) to the darker ones (marked by 0).

Outputs coming from a convolutionl layer that has two filters that detect both vertical and horizontal edges are like those in figure 2.26.

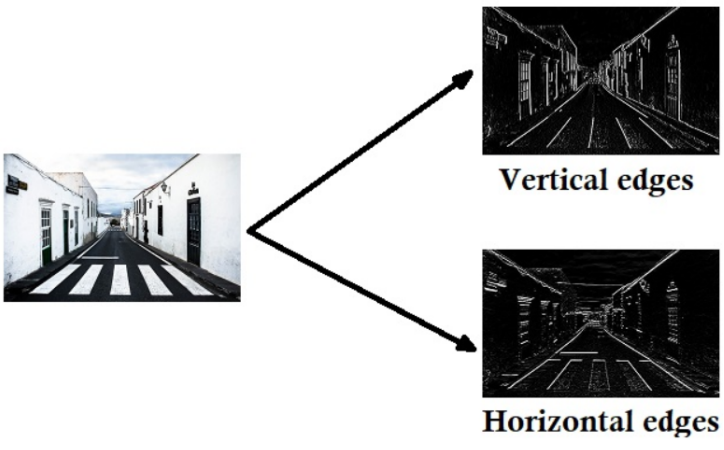


FIGURE 2.26: Vertical and horizontal edge detection. *Source: datahacker.rs* [14]

In CNNs, parameters of each feature map are not fixed, so rather than detecting necessarily vertical or horizontal edges, the network can learn to detect edges at whatever orientation, adapting the filter to image features.

3D convolutions

When features are detected in RGB images, *3D convolutions* or *convolutions over volume* are performed.
The input image is now described by a matrix on each of the three different channels: red, green and blue. The number of channels is identified by the last dimension of the image, after its height and width.
The feature map must have channels whose number matches the one of the input matrix, so here it concides with three. The filter can be seen just like a (yellow) little cube that slides along the entire input image.

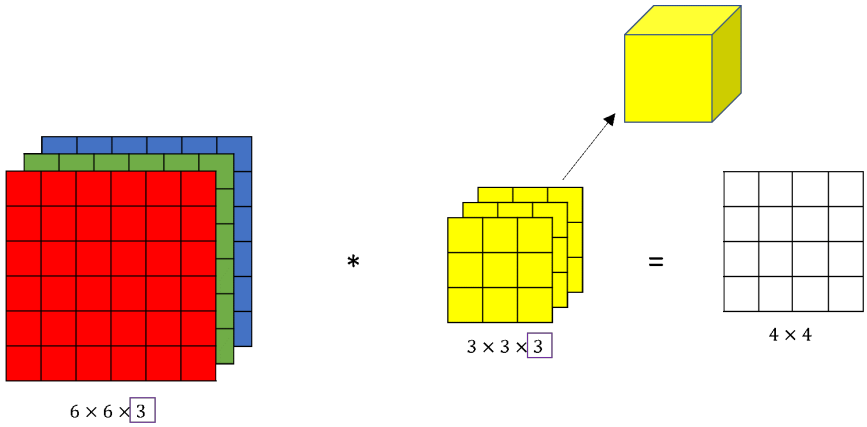


FIGURE 2.27: Convolution over volume. *Source: datahacker.rs* [14]

The 3D convolution consists of multiplying all numbers belonging to the three faces of the cube by the corresponding numbers from the red, green and blu channels, adding these products and then applying an activation function.
The resulting number is inserted in the appropriate square of the output image.
In the picture 2.27, the 6×6×3 image is convolved with a 3×3×3 filter.
Pay attention to the output dimensions: a 2D matrix of 4×4 pixels comes out (m-f+1=6-3+1=4).

Zero padding

The tecnique called *zero padding* modifies the basic convolutional operation.
The original image (the green one in picture 2.28) is padded with an additional border of zeros around the margins, before performing convolutions.
The purpose of padding is twofold: the image is no more shrinked every time edges are detected, preserving the original input size, and pixels near the margins are considered more than before, enhancing the information they bring.

2.4.2 Pooling layers

The *pooling layers* are usually present after convolutional layers. They are added to reduce the size of the representation, to speed up computations and, mostly, to make the detected features more robust.

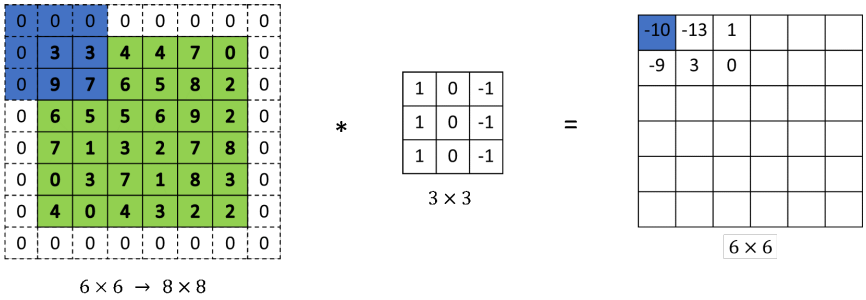


FIGURE 2.28: Padding an image. *Source: datahacker.rs* [14]

There are two types of pooling: *max pooling* and *average pooling*. In both, there are no parameters to learn becuse the computation is fixed.

Max pooling This tecnique cuts the image in regions and takes the maximum number inside each one. A large number inside a region means that a particular feature is detected, so in this way we reach some flexibility in finding features, accounting for any distortions. In the image 2.29 below, a 4×4 input image is shrinked into a 2×2 image using a 2×2 filter and stride equal to 2.

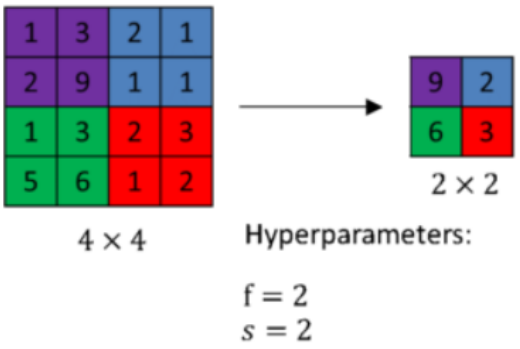


FIGURE 2.29: Max pooling. *Source: datahacker.rs* [14]

Average pooling This tecnique, instead, makes the average of numbers inside each region. It is used only when we want to collapse the representation. In the image 2.30, the output is still a 2×2 image, but the average pooling is used.

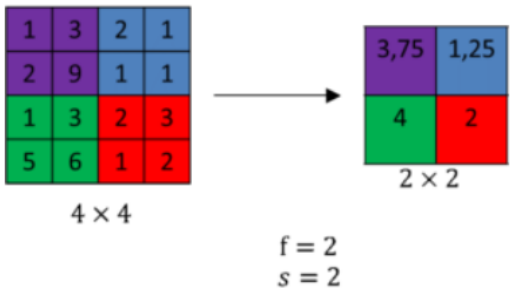


FIGURE 2.30: Average pooling. *Source: datahacker.rs* [14]

2.4.3 Fully connected layers

Usually, the output coming from the last feature map is flattened into a vector and connected to some *fully connected layers*.
It is like adding an Artificial Neural Network in the final part of the network, whose role is to interpret the extracted features.

2.4.4 Example of a CNN architecture

In the figure 2.31, the architecture of *LeNet-5* is shown.
This is one of the erliest Convolutional Neural Netwoks, proposed in 1998 by Yann LeCun et al. at Bell Labs. The goal of LeNet-5 is to recognize handwritten digits, after a training with grayscale images.

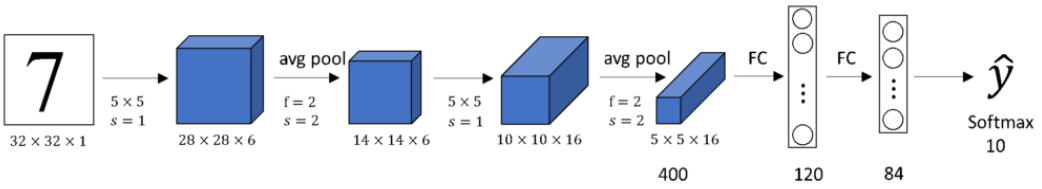


FIGURE 2.31: Architecture of LeNet-5.*Source: datahacker.rs* [14]

The network takes as input a 32×32×1 image, that is firstly processed by a convolutional layer with 6 5×5 filters, using a stride length of 1.
The resulting 28×28×6 volume is later shrinked by an average pooling layer with hyperparameters f=2 and s=2, outputting a 14×14×6 cube.
Then, an identical block of convolutional-pooling layers is present, producing a 5×5×16 volume.
Notice that the number of channels increases as we are closer to last convolutional layers, while the height and the width become smaller.
The volume is finally flattened into a vector of 400 units, that are linked to two fully-connected layers with 120 and 84 neurons, respectively.
The network ends with a softmax layer that classifies the handwritten digit among the ten possible classes.

Chapter 3

OCC State of Art

The objective of this work of thesis is the development of a *One-Class Classification* (OCC) algorithm where the *person class* is the target class.
In this chapter, firstly, an investigation into general concepts of *classification* is made, highlighting differences with other tasks like regression.
Later, the *classification of a single class* is defined and explored in contrast to binary and multi-class classifications.
Finally, various *applications* of One-Class Classification are presented, followed by a roundup of *popular approaches* used to realize them.

3.1 Classification inside Machine Learning

The *classification* is a task belonging to *supervised learning algorithms*, where both inputs and desired outputs are available during the training.
The goal of a classification algorithm is to identify the category of which new data are part, by relying on observations available in the training set.
Considering the generic problem

$$y = f(x),$$

we can identify x as the input, f as the model and y as the output.
The output of this kind of problems is a discrete value, known as *label*, *category* or *class*.
Therefore, the mapping function f links input data to categorical or boolean values and is referred as the *classifier*.
In the event that no labels are provided in the problem formulation, we are faced with a different case named *clustering*, that is the corresponding unsupervised algorithm of classification. Here data are grouped into categories without knowing a priori the classes.

3.1.1 Classification and regression

Classification differs also from *regression*, another important prediction problem. In the latter y is a continuous real value and represents a numerical quantity that cannot be interpreted as a category.
A practical example to clarify the two tasks is shown in figure 3.1.

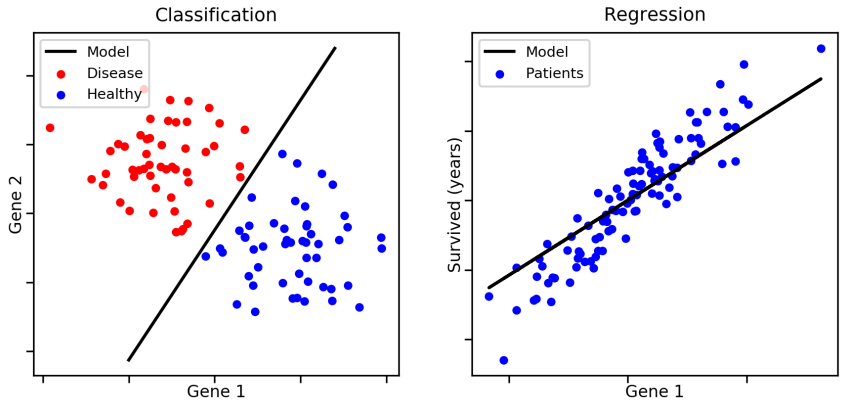


FIGURE 3.1: Classification and Regression. *Source: aldro61* [15]

Classification (on the left) categorizes patients into two classes, healthy or diseased, on the basis of genomic measurements of gene 1 and gene 2. The black line separating red and blue dots is the *decision boundary*, which we look for. Regression, instead, predicts for how many years a patience survives, having a specific genomic measurement of gene 1. Here, the black line among blue points, representing patients, is the *best fit line*: it is found to precisely predict the quantity of years.

Other examples of common classification problems are:

- the spam filter that automatically classifies new emails;
- image classification in manufacturing lines to identify products;
- image classification in medical field to detect cancer cells;
- text classification in order to categorize new articles.

3.1.2 Image classification in Computer Vision

In this work of thesis, a One-class Classification algorithm is developed to identify instances of people. Our purpose is to understand whether people are present in the images or not.
This task is defined in *computer vision field* as *Image Classification*.
Attention shall be paid to not confuse image classification with other computer vision concepts, which seem similar but that are heavily different in practice: *Image Localization*, *Object Detection* and *Image Segmentation*.
Hereafter, more detailed descriptions are presented to make these methods unambiguos.
In *Image Classification* the content of an image is identified; this tecnique tells which things are present inside it. Questions like "What is in the picture?" or "Is this a cat or is it a dog?" are answered.
In *Image Localization*, as suggested by the term, the localization of an object is provided in addition to the type of the object.
Object Detection is, instead, the generalization of image localization, used when

several objects are present in the image. It detects the objects drawing around them rectangles or squares, called *bounding boxes*. It provides the information of *'where are they'* in addition to *'what are they'*.
Finally, *Image Segmentation* cuts the pictures into segments and groups together pixels with similar features. The result of this method is that each object is characterized by a pixel-wise mask, identifying all shapes of different objects.
The four methods can be better understood looking at figure 3.2.

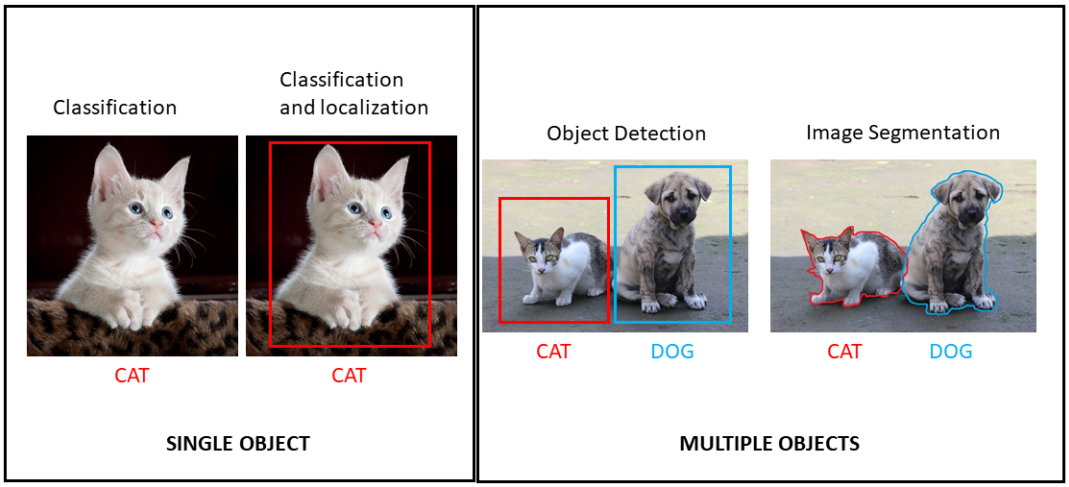


FIGURE 3.2: Classification, Localization, Object Detection, Segmentation. *Source: natasa* [16]

3.2 One-Class Classification

It's time now to approach the centerpiece of our investigation: *One-Class Classification*, also called *Single-Class Classification*.

3.2.1 The goal of OCC

The term is used for the first time in 1975 by T.C. Minter in the design of a Bayes classifier, that is actually the first semi-supervised One-class classifier.
In his article *"Single-Class Classification"*, data are categorized into two classes, the *class of interest* and the *other class*, but labeled samples only from the first class are demanded during the training.
Minter states:

> "[...] a Bayes classifier will be presented which classifies samples into the "class of interest" or the "other" classes but requires only labeled training samples for the "class of interest" to design the classifier. Thus, this classifier minimizes the need for ground truth. For these reasons, the classifier will be referred to as a single-class classifier".

In 1993, Moya and Hush give a later and concise definition of a One-Class classifier in the research paper titled *"Network constraints and multi-objective optimization for one-class classification"*:

> "We call a classifier that can recognize new examples of target patterns and distinguish those from non-target patterns a one-class classifier".

In recent years (2004), Piotr Juszczak in the work *"Combining one-class classifiers to classify missing data"* asserts:

> "In the problem of one-class classification the goal is to accurately describe one class of objects, called the target class, as opposed to a wide range of other objects which are not of interest, called outliers".

After clarifications up there, the goal of the One-Class Classification is evident: in this problem, objects of a particular class are identified compared to all possible other objects.
The particular class is called *positive class* or *target class*, while other items are referred to be in the *negative class*, *alien class*, or depending on the application, in the *outilier*, *intruder* or *novel class*.
The peculiarity of OCC arises strong during the training: only instances of the same positive class are available, so just one of the classes is well characterized. The outlier class has very few objects, no objects at all, or simply the negative concept cannot be represented by samples we dispose.
The difficulty of the One-Class Classification is, therefore, inside the lack of information coming from the alien class.
This kind of problem is harder than standard methods beacuse the classification boundary, being defined on the basis of just one class, is difficult to place and, also, because of the definition of the features that characterize positive instances and that guarantee the right distance from negative ones.

3.2.2 One-class versus binary classification

One-Class Classification differs from binary (or multi-class) classification, precisely because in the latter the training set has labeled data from all the predefined classes. This exhaustive dataset is missing in the OCC: no instances of a second class (or multi-classes) are available during the training, so objects of the class of interest are categorized using solely instances of the same class.
Moreover, since classical classification problems create discriminatory functions basing on instances from all classes, they are naturally said to be discriminatory. This property requires an equally balanced dataset, in order to build an effective decision boundary.
When examples of a class are much more abundant than others, discriminatory methods may not perform very well and can not be employed. Consequently, One-Class classifiers emerge.
Real situations in which data from only one class are available, bacause alien objects are absent or in a limited quantity, are very common and will be treated

in next section.
In summary, if the dataset is structured with a sbalanced number of examples from all classes, a binary (or multi-class) classification is recommended.
On the contrary, in unbalanced cases when an abundance of examples of a particular class is seen, One-Class Classification is the solution.

3.2.3 Applications of One-Class Classification

It is now natural to ask in what kind of situations unbalanced datasets are indispensable. Some examples are presented below:

- in *typist recognition* a huge number of instances from the alien classes are present. In this scenario we can't have all possible cases inside the training set;
- in *machine faults detection*, for example, during the monitoring of helicopter gearboxes, in the operational status check of a nuclear plant or during the detection of oil spills. Here, the negative class includes all possible abnormal behaviours, but primarily they are very rare and, then, they may cause risks to people and also result in high costs.
  Waiting for the occurrence of faults is not a good strategy. Building a One-Class classifier according to normal observations of the machine is, instead, a solution;
- the *automatic diagnosis of a disease*: positive data are represented by "common" diseases, easy to group together, while negative class is formed by "rare" ones. The outlier category is difficult to fill because tests on rare deseases are very expensive and patients are uncommon, making the negative class poor of instances;
- in *homepage classification* or in the *inclusion of journal articles* for systematic reviews is very easy to collect positive instances, while it is difficult to group together all negative examples. The negative class may not be uniformly represented and, also, the categorization may be altered by subjective choices;
- in *mobile active authentication*, we dispose images only from the current user, since objects of the negative class, the other users, are hard to collect due to privacy issues.

In these scenarios, data from only one class are available, given that instances of the negative class are very hard/impossible to have, otherwise the negative concept is not fully represented by our samples.

All these situations can be grouped in three main computer vision applications: *novelty detection*, *anomaly detection* and *mobile active autentification*.
In *novelty detection*, the objective is to find novelties with respect to observed examples. It is therefore normal that data from novel class are not known, it would be counterintuitive.

The goal of *anomaly detection* is to identify abnormal data. Since training is done using normal operating examples, our classifier has to learn the concept of normality.
In *mobile active autentification*, the identity of a user is constantly verified. Only his examples are available to discriminate negative instances.

3.3 Popular approaches in OCC

There are several approaches in literature for solving One-Class Classification problems. Main tecniques are presented in this section.

3.3.1 One-Class Support Vector Machines

In the article *"Support Vector Method For Novelty Detection"* [17], Schölkopf et al. propose a method that extends the concept of *Support Vector Machines* to problems where unlabeled data are present, in short, OCC problems.
In order to clarify the arising *One-Class Support Vector Machines*, basics of Support Vector Machines are first shown.

Support Vector Machines (SVMs) are supervised learning algorithms used mostly for classification. The typical SVM that presents two classes is analyzed.
Given a training set of n labeled examples, the model separates points of different classes finding a boundary that maximizes the gap between them. New observations will be assigned to a certain category or to the other one, basing on the side of the gap they are within.
When data are not linearly separable, a straight boundary is not able to divide them and the so-called *"kernel trick"* is used: data are projected into higher dimensional space, called *feature space*, through a non-linear function ϕ.
In figure 3.3, blue points cannot be separated from red ones through a straight line (2D) and they are, therefore, moved to a feature space where a hyperplane is used for the purose (3D). When we come back to 2D space, the straight line becomes curve.

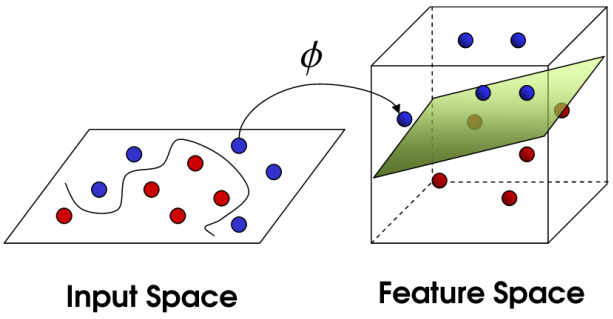


FIGURE 3.3: Mapping to higher dimension in SVM. *Source: D. Wilimitis* [18]

The goal of SVMs is to find the hyperplane that maximizes the distance between the closest points of the two classes. The problem can be reformulated in terms of minimization in this way:

$$\min_{w,b,\xi_i} \quad \frac{||w||^2}{2} + C\sum_{i=1}^{n}\xi_i \tag{3.1}$$

$$\text{s.t.} \quad y_i(w^T\phi(x_i) + b) \geq 1 - \xi_i \quad \forall i = 1, ..., n \tag{3.2}$$

$$\xi_i \geq 0 \quad \forall i = 1, ..., n. \tag{3.3}$$

In particular, parameters w and b belong to the equation of the hyperplane $w^T x + b = 0$, x_i are generical points of the dataset to whom y_i outputs are associated, ξ_i are slack variables to have a soft margin and C is a constant value that balances margin maximization and the number of points inside the margin.
Solving the above quadratic programming problem with Lagrange multipliers leads to:

$$f(x) = sgn(\sum_{i=1}^{n}\alpha_i y_i K(x, x_i) + b),$$

where $f(x)$ is the *decision function* of a generic point x, α_i are the *Lagrange multipliers* and the term $K(x, x_i) = \phi(x)^T\phi(x_i)$ is the *kernel function*.
The kernel function maps points in higher dimensional spaces and can be linear, polynomial or also the popular *gaussian radial base function*:

$$K(x, x_i) = exp(-\frac{||x - x'||^2}{2\sigma^2}),$$

with σ, real number that is the kernel parameter, and $||x - x'||^2$, the dissimilarity measure.
The two-class Support Vector Machine can be now applied to data only belonging to one class, resulting in One-Class Support Vector Machine.

The method proposed by Schölkopf et al. consists in the separation of all observations from the origin in the higher dimensional space, realized thanks to the hyperplane. The feature space will be therefore divided into areas, each characterized by probability densities of points.
While in binary SVMs the value *-1* is associated to one side and the value *1* to the other one, in OCSVMs the region where data points lie is marked with *+1*, while all other parts return *-1*.
The goal of SVMs in OC problems is to maximize the distance between the hyperplane and the origin.
The resulting minimization problem is close to (3.1) formulation:

$$\begin{aligned} \min_{w,b,\rho} \quad & \frac{||w||^2}{2} + \frac{1}{\nu n}\sum_{i=1}^{n}\xi_i - \rho \\ \text{s.t.} \quad & (w \cdot \phi(x_i)) \geq \rho - \xi_i \quad \forall i = 1, ..., n \\ & \xi_i \geq 0 \quad \forall i = 1, ..., n. \end{aligned}$$

The parameter $\nu \in (0, 1]$ is very important because determines both an *upper bound* for out-of-class points with respect to all data points and a *lower bound* on the number of observations from the training set.
The quantity ρ, instead, is the distance between the origin and the hyperplane.
The decision function is now:

$$f(x) = sgn(\sum_{i=1}^{n}\alpha_i K(x, x_i) - \rho).$$

In summary, since data from the negative class are missing, the origin is treated as the outlier part of the feature space: the hyperplane maximizing the distance of the target class from the origin is found.

3.3.2 Support Vector Data Description

Another famous approach is described in the article *"Support Vector Data Description"* [19]. Inspired by OCSVMs, Tax et al. suggest to find a spherical boundary to divide data points in the higher dimensional space, rather than using the hyperplane. This kind of boundary is called *hypersphere* and is characterized by a center c and a radius $R > 0$.
The goal of SVDD is to minimize the volume of the hypersphere that contains the target class data, embedding the fewest outliers.
The above formulation can be rewritten as the following minimization problem:

$$\begin{aligned} \min_{R,c,\xi} \quad & R^2 + \frac{1}{\nu n}\sum_{i=1}^{n}\xi_i \\ \text{s.t.} \quad & ||\phi(x_i) - c||^2 \leq R^2 - \xi_i \quad \forall i = 1, ..., n \\ & \xi_i \geq 0 \quad \forall i = 1, ..., n. \end{aligned}$$

The parameter $\nu \in (0, 1]$ determines the trade-off between the volume of the hypersphere and slack variables ξ_i, that allow to have a soft margin.
A new observation is said to be in-class when $||\phi(x) - c||^2 \leq R^2$; on the contrary, if a new point falls outside the hypersphere it is considered an outlier.

In figure 3.4 the two approaches, One-Class Support Vector Machine (left) and Support Vector Data Description (right) are illustrated. Despite charts are in 2D, they show the feature spaces. All empty dots are outliers, divided from target class points by the hyperplane and by the hypersphere.

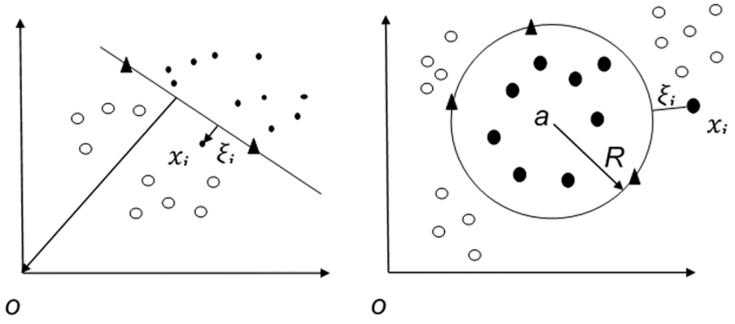


FIGURE 3.4: One-Class Support Vector Machines and Support Vector Data Description. *Source: Vasighizaker et al.* [20]

3.3.3 One-Class Classification with Gaussian Processes

In the article *"One-Class Classification with Gaussian Processes"* [21], Gaussian Process (GP) priors are employed in One-Class Classification problems. First of all, a clarification on GP classification is provided, starting from the simpler regression case.

Given a training set of n labeled examples, the mapping between $\mathbf{X} = [x_1, x_1, ..., x_n]$ and $\mathbf{y} = [y_1, y_2, ..., y_n]$ is searched. The latent function f and the additive noise ϵ can link them: $\mathbf{y} = f(\mathbf{X}) + \epsilon$.
In the *Bayesian approach*, the mapping function f has not a predefined structure tied to certain parameters: it is described by a probability distribution $p(f|\boldsymbol{X})$ that is updated when new observations are available.
Considering a new data point x_*, the probability of its output y_* is based on previous data $\mathbf{X}$ and $\mathbf{y}$, updating function values $f_* = f(x_*)$ and $f = [f(x_1), ..., f(x_n)]^T$:

$$p(y_*|\mathbf{X}, \mathbf{y}, x_*) = \int p(f_*|\mathbf{X}, \mathbf{y}, x_*)p(y_*|f_*)df_* \tag{3.4}$$

$$p(f_*|\mathbf{X}, \mathbf{y}, x_*) = \int p(f_*|\mathbf{X}, \mathbf{f}, x_*)p(\mathbf{f}|\mathbf{X}, \mathbf{y})d\mathbf{f}. \tag{3.5}$$

In *Gaussian Process regression*, there is a strong hypothesis: all functions $\mathbf{f}$ are jointly normally distributed. It follows that each distribution can be described by mean $m(\cdot)$ and covariance $\kappa(\cdot, \cdot)$ functions:

$$\mathbf{f}|\mathbf{X} \sim \mathcal{N}(m(\mathbf{X}), \kappa(\mathbf{X}, \mathbf{X})).$$

If we use a zero-mean Gaussian Process, the formulation of the probability distibution (3.5) is still a Gaussian having:

$$\mu_* = k_*^T(\mathbf{K} + \sigma_n^2\mathbf{I})^{-1}\mathbf{y}$$
$$\sigma_* = k_{**} - k_*^T(\mathbf{K} + \sigma_n^2\mathbf{I})^{-1}k_*,$$

where $\mathbf{K}$= $\kappa(\mathbf{X}, \mathbf{X})$, $k_* = \kappa(\mathbf{X}, x_*)$, $k_{**} = \kappa(x_*, x_*)$.
Additionally, when ϵ is assumed to be an independent and identically distributed

Gaussian noise, integrals can be solved in closed form and the probability of output y_* in (3.4) is normally distributed too, characterized by mean μ_* and variance equal to $\sigma_*^2 + \sigma_n^2$.

In GP regression, y are real numbers. Let's move to *GP classification* where y consist in discrete numbers, rapresenting the binary labels.
Equations (3.4) and (3.5) with Bayesian approach can be used, but the hypotesis on Gaussian noise is no more valid beacuse $y \in \{-1, 1\}$.
We can employ a different likelihood like the cumulative Gaussian distribution

$$p(y|f) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{yf} exp(-\frac{1}{2}x^2)dx, \tag{3.6}$$

but we lose the advange of $p(f_*|\mathbf{X}, \mathbf{y}, x_*)$ to be normally distributed.
A solution for the equation (3.4) is to consider the posterior probability $p(\mathbf{f}|\mathbf{X}, \mathbf{y})$ as a normal distribution $\hat{p}(\mathbf{f}|\mathbf{X}, \mathbf{y})$, through *Laplace approximation* or *Expectation Propagation*.
Finally, considering Gaussian approximations in (3.5) and the already defined $p(y|f)$ in (3.6), the probability $p(y_*|\mathbf{X}, \mathbf{y}, x_*)$ can be seen as a cumulative Gaussian distibution.

It's time now to extend the concept of Gaussian Process Priors to One-Class Classification.
The equation in (3.5) describes a conditional probability density, which means that GP classification works in a discriminatory way and so, it is not easy to adapt to OCC problems where density of input data is missing.
However, scores to identify target class elements can be found if we choose correctly the Gaussian process prior: if we select a mean of the prior equal to zero, i.e. a value lower than the positive class labels ($y = 1$), probable latent functions will have values that become smaller moving away from available data points. Several appealing latent functions for One-Class Classification can be picked, also having a soft covariance function. This implies that the probability $p(y_* = 1|\mathbf{X}, \mathbf{y}, x_*)$ can be used in OCC, even if it is discriminative, and also mean and variance can be further membership indicators.
The following table 3.1 resumes quantities extracted from the predictive distribution that can be used as membership scores in One-Class Classification.

Mean (M)	**Negative Variance** (V)	**Probability** (P)	**Euristic** (H)
$\mu_* = \epsilon(y_*\|X, y, x_*)$	$-\sigma_*^2 = -\nu(y_*\|X, y, x_*)$	$p(y_* = 1\|X, y, x_*)$	$\mu_* \cdot \sigma_*^{-1}$

TABLE 3.1: Measures for OCC membership scores. *Source: Kemmler et al.* [21]

In particular, while the mean M can be used without modifications because it decreases for the outliers, the variance V is taken negative since it increases.

3.3.4 Single-Class Minimax Probability Machines

Another approach for One-Class Classification is shown in the article *"Robust novelty detection with single-class MPM"* [22].
Lanckriet et al. use the *Minimax Probability Machine* to find appropriate boundaries around the training set in the task of novelty detection.
The goal is to discover a minimal region Q in the input space that holds the fraction α of the probability mass of the dataset:

$$\mathbf{Pr}\{\mathbf{y} \in Q\} = \alpha.$$

In novelty detection $\alpha \in (0, 1]$ is usually set closer to 1.
This formulation is know as *quantile estimation*, in which MPMs are suitable since they are characterized by the following theorem:

$$\inf_{y \sim (\bar{y}, \Sigma_y)} \mathbf{Pr}\{a^T y \leq b\} \geq \alpha \iff b - a^T \bar{y} \geq \kappa(\alpha)\sqrt{a^T \Sigma_y a},$$

in which a and b are constants (a different from 0, $a_T \bar{y} \leq b$) and $\kappa(\alpha) = \sqrt{\frac{\alpha}{1-\alpha}}$.
The *infimum* over various distributions of y is computed, with the only constraints of mean equal to $\bar{y}$ and covariance matrix equal to Σ_y.
The research of the optimal region $Q(\gamma, b)$ in One-Class case, if the choice of α is feasible, is handled through a second-order cone programming problem:

$$\begin{aligned} \min_{\gamma} \quad & \sqrt{\gamma^T (L^T L + \rho K)\gamma} \\ \text{s.t.} \quad & \gamma^T k - 1 \geq (\kappa(\alpha) + \nu)\sqrt{\gamma^T (L^T L + \rho K)\gamma}, \end{aligned}$$

in which $\kappa(\alpha) = \sqrt{\frac{\alpha}{1-\alpha}}$, b is equal to 1, K is the Gram matrix.
Also, elements of vector k are $[k]_i = \frac{1}{N} \sum_{j=1}^{N} K(x_j, x_i)$, where x_i are the N observations, and $L = \frac{K - 1_N k^T}{\sqrt{N}}$.

Alternatively, the following result:

$$\begin{aligned} \gamma_* &= \frac{(L^T L + \rho K)^{-} 1k}{\zeta^2 - (\kappa(\alpha) + \nu)\zeta} \\ \text{with} \quad \zeta &= \sqrt{k^T (L^T L + \rho K)^{-} 1k} \end{aligned}$$

is used to find the optimal half-space, given $\alpha \leq \frac{(\zeta - \nu)^2}{1 + (\zeta - \nu)^2}$ or $\kappa(\alpha) \leq \zeta$.

If there are no feasible values for α, no solution for the problem can be found.
The complexity of both formulations is comparable to the one of OCSVM, since it reaches $O(N^3)$ in worst-cases.

3.4 One Class-Classification with CNNs

In last years, methods basing on deep *Convolutional Neural Networks* have been proposed for OCC problems, since they achieve excellent performances in very complex applications.
Computer Vision is also a breeding ground with huge amount of labeled examples from different areas.
In One-Class Classification problems, CNNs try to extract features that characterize the particular class, while the lack of information coming from the negative class is filled in different ways.
A brief description of two popular methods are here presented.

3.4.1 Deep Support Vector Data Description

Ruff at al. propose to use Deep Learning tecniques in the task of *anomaly detection* in the article *"Deep One-Class Classification"* [23]. They put together the method of Support Vector Data Description and the framework of Neural Networks, giving rise to *Deep Support Vector Data Description* (figu. 3.5).

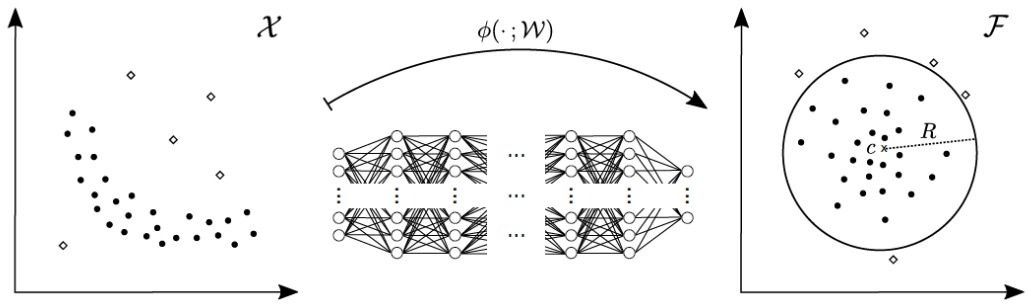


FIGURE 3.5: Deep Support Vector Data Description. *Source: Ruff et al.* [23]

In Deep SVDD, a neural network is trained whereas it tries to minimize the volume of the hypersphere with inside the target data points. Hence, the goal is to learn the transformation $\phi(\cdot; W)$ that maps points from the input space X to the feature space F, maximizing examples inside the spherical boundaries. Outliers will lie outside the volume, far from its center c.

Considering a training dataset of n examples $D_n = \{x_1, ..., x_n\}$ and a Neural Network with L hidden layers and weights $W = \{W^1, ..., W^L\}$, the objective of Deep SVDD for OCC is:

$$\min_{W} \quad \frac{1}{n}\sum_{i=1}^{n} ||\phi(x_i; W) - c||^2 + \frac{\lambda}{2}\sum_{l=1}^{L} ||W^l||_F^2.$$

The first quantity represents the quadratic loss involving distances in the feature space of points from c, while the second one is a regularizing term, tied to the hyperparameter λ.

3.4.2 One-Class Convolutional Neural Network

In the article *"One-Class Convolutional Neural Network"* [24], Oza et al. propose the following architecture for OCC problems (figure 3.6).

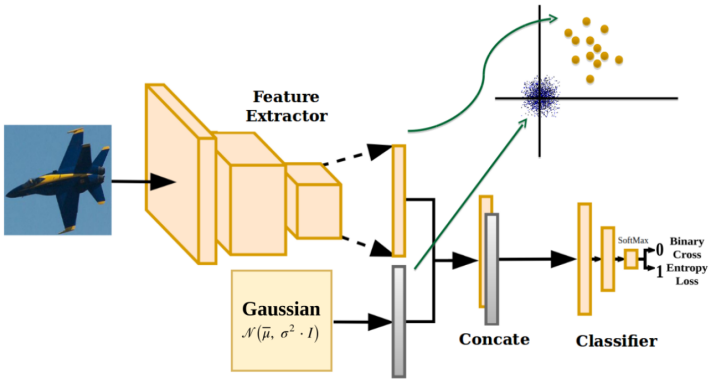


FIGURE 3.6: One-Class Convolutional Neural Network. *Source: Oza et al.* [24]

The first block of the model is a feature extractor realized through a pre-trained CNN. It is fed by target class images and it extrapolates the distinctive features. The negative objects come from a fictitious class generated by a zero centered Gaussian noise with mean $\bar{\mu}$ and standard deviation σ.
The extracted features from both classes are appended and, then, sent to a last block of fully connected layers, that classify the instances as 0 or 1 thanks to some confidence scores.

3.4.3 Other approaches

Other approaches in solving OCC problems are the so-called *generative approaches*, that employ generative models like *auto-encoders* and *Generative Adversarial Networks* (GANs).

Examples of these solutions can be found in the following articles:

- *"Finding anomalies with generative adversarial networks for a patrolbot"* by Lawson et al. [25];
- *"Abnormal event detection in videos using generative adversarial nets"* by Ravanbakhsh et al. [26];
- *"Fully convolutional neural network for fast anomaly detection in crowded scenes"* by Sabokrou et al. [27].

Chapter 4

Deep One-class Classification

In this chapter, the method used for the development of the people recognition algorithm, called *Deep One-class Classification*, is studied.
This solution for One-Class Classification is proposed by Perera and Patel in the article *"Learning Deep Features for One-Class Classification"* [28], published in November 2019.
The structure of the chapter is the following: first of all, some key concepts of the method are explained, in particular what *feature learning* means and what is, instead, the *transfer learning*. Then, common frameworks used in Deep Learning, including the used framework in DOC, are shown.
Finally, the *DOC optimization problem*, the two *misurable loss functions* and the *training architecture* are presented.

4.1 Fundamental concepts in DOC

In this section, let us to be clear some important points, on which Deep One-class Classification is based.

4.1.1 One-Class feature learning

The key element in the discussion is *learning deep features* that characterize instances of people.
It is not feasible to acquire all possible counter-examples different from objects of the target class, that is the reason why a multi-class classification model or a binary model cannot be used.
The algorithm has therefore to understand what kind of properties to isolate in order to distinguish a person from anything else.
The class composed by person examples is highly wide: people can be in the foreground or far from the camera, they are of different ages (kids, adults, elders) and different gender (male, female), they may have various physical traits, skin colors, clothes, and also disparate poses: pictures of individuals are taken from the back, from di front, on the side.
Features are different and heterogeneous, this is very challenging.

The suggested method employs a Convolutional Neural Network to extract these high-level features: during the training, person images pass through several convolutional and pooling layers and the network learns filter parameters, as

described in section 2.4.
The computation of the customized loss function called *compactness loss* helps in this process, evaluating the compactness of person class in the feature space.

4.1.2 Transfer learning

Another key element in the treatment is the concept of *transfer learning*.
This tecnique allows to adapt a labeled dataset from an independent task to the One-Class Classification.
In our problem all classes different from the person class are referred to be *alien classes* and objects that belong to them, *alien objects*. Since we cannot create an exhaustive negative class with all examples opposed to people, an *external multi-class dataset* is employed during the training, besides the *one-class dataset* with people images. The *descriptiveness loss* evaluates the descriptiveness of the features extracted from external images.
The multi-class dataset is called also *reference dataset* and it is an arbitrary collection of images available online, like *ImageNet*, *ILVRC12*, *Places35* datasets or a subset of them.
In our implementation, the reference dataset is a subset of 10000 images from *ILVRC12 dataset*.

4.2 Deep Learning-based frameworks

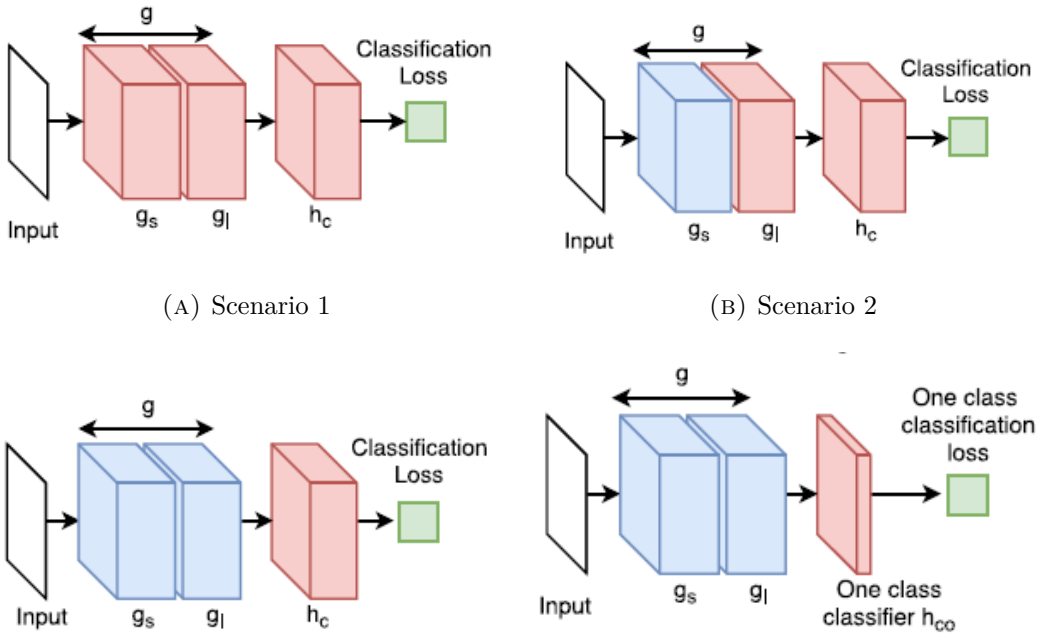
Usually, in Deep Learning-based frameworks for classification there are two blocks: a network for *feature extraction*, referred as g, and a *classification part*, called h_c, after which loss functions to be minimized are computed using the output coming from h_c.
Depending on the number of classes and on the number of objects per category, different scenarios may occur. Some of them are described below to justify the strategy used in Deep One-class Classification:

- **scenario 1** *several classes and large number of training examples:* in this situation there are enough data to train all parts of the model in a end-to-end fashion. Both g and h_c learn from scratch, because all parameters are randomly inizialized and they are adjusted during the training to reach a final goal;

- **scenario 2** *several classes and medium number of training examples:* in this case the feature extraction network is cut into two parts, the *shared feature network* g_s and the *learned feature network* g_l. The first block g_s, colored in blue, is said to be *frozen*, since all the weights and biases associated to its layers are not updated during the training. Parameters are imported from a pre-trained model, taking the low-level features learned in a different problem with an external dataset. Subsequent blocks g_l and h_c (in red) are instead trained end-to-end, keeping information from new available data;

- **scenario 3** *several classes and small number of training examples:* here, the entire feature extraction block g is *frozen*, because there are no enough examples for learning consistent features. It can be a pre-trained model, coming for example from scenario 1. The classifier takes the extracted high-level features and has the role of learning how correctly discriminate objects from different classes;
- **scenario 4** *one class training examples:* in this situation the first part is completely *frozen* too and parameters are taken from a pre-trained model from scenario 1 or scenario 2. The classification block is a one-class classifier that learns through a one-class dataset how dealing with extracted features from g.

In figure 4.1, the four possible situations are shown: white parts are input images fed in the network, red blocks are parts that learn during the training, blue blocks are *frozen* and the final green block is where the classification loss is evaluated.



(A) Scenario 1

(B) Scenario 2

(C) Scenario 3

(D) Scenario 4

FIGURE 4.1: Different scenarios in DL-based frameworks. *Source: Perera and Patel* [28]

4.2.1 DOC framework

In the Deep One-class Classification, *scenario 4* is considered since only instances from a single class are available.
If the latter framework is used without any modifications, it can happen that deep features coming from g are not valid also in our One-Class Classification problem. This is because the frozen model is previously trained using a dataset that differs from the current one and, consequently, features are taken to fulfil

the previous task.
After making some appropriate changes, a new framework is proposed for DOC (in figure 4.2).

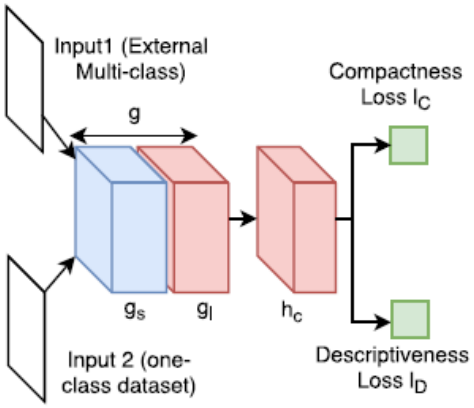


FIGURE 4.2: New framework for DOC. *Source: Perera and Patel* [28]

The big advantage of using this model is that now the extracted features are consistent with our task, that is *One-Class novelty detection* applied to the *person class*.
There is still a frozen part in blue, the shared feature network g_s, which produces the low-level features, that are mantained because general. Red blocks g_l and h_c have to learn during the training the specialized features, thanks to the *compactness loss*, computed using one-class examples to force the person class to be compact in feature space, and the *descriptiveness loss*, computed using the multi-class dataset to maximize the inter-class distance among classes in feature space.
In this way, we monitor the quality of the deep features coming from g, now suitable for describing the person class compared to anything else.

4.3 DOC optimization problem

It is necessary to understand how specialized features for Deep One-class Classification are produced by the *feature extraction block*, defining a consistent *optimization problem* to solve.
We cannot follow the solution found for traditional classification problems: discriminatory functions can't be realized because only samples from the target class are available and there are no multiple classes.
We focus, instead, on two main properties that features in OCC problems shall have and, later, we try to find some mathematical functions that incorporate them.

The first one is the *compactness*. Different instances of the same category should have similar features, that identify the class itself. The representation of features extracted by objects of a class is a compact cloud in feature space: all

points, each of them corresponding to one instance, are localized in a dense way.

The second property is the *descriptiveness*. Objects of different categories should have different features. The representation of features coming from instances of different classes is placed away from others.
Also in multi-class classification, both compactness and descriptiveness are fundamental qualities for extracted features, described by a low intra-class distance and a large inter-class distance, respectively.
Picture 4.3 gives the idea of the intra-class and the inter-class distances in a three-class classification problem.

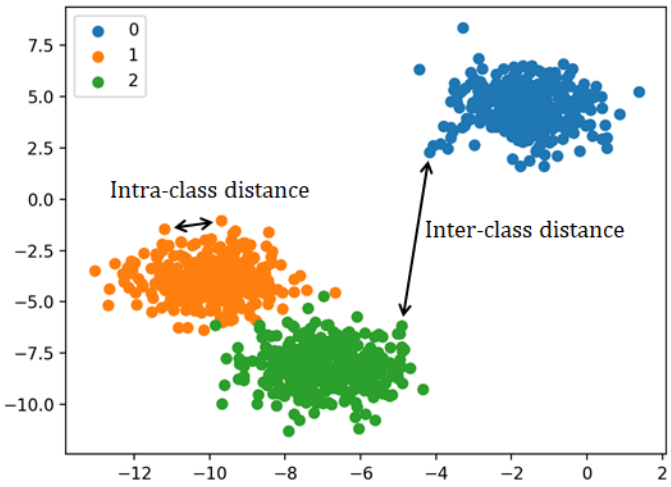


FIGURE 4.3: Intra-class and inter-class distances. *Source: J. Brownlee* [29]

Therefore, a feature is considered effective for Deep One-class Classification if it benefits from compactness and descriptiveness at the same time.
Our purpose becomes to look for a feature extractor g able to make these quantities bigger. Considering the one-class training dataset as t, the following *optimization problem* can be written:

$$\hat{g} = \max_{g} D(g(t)) + \lambda C(g(t)).$$

The first term refers to the descriptiveness D of features extracted by g, having t as input of the network, the second part of the equation is the compactness C of the same features, while λ is a constant that allows to correctly weight the two quantities.

4.4 Misurable loss functions

Realized what are requirements of specified features for person classification task, it's time to define some mathematical functions to measure levels of compactness and descriptiveness.

A distibution is said to be compact if it is not widespread. The distribution spread is usually assessed by means of *variance*. If we want compactness in features, the way forward is clear: we have to minimize the variance of their distribution.
Due to memory limitations and acceptable time needed for simulations, it is not possible to consider simultaneously all target examples we have at our disposal: at each step only groups of people images are fed into the network, called *batches* of images. Since features related to a single batch are available, the variance of batch features plays the role of the variance of the entire feature distribution. In this way, the variance of features of each batch, referred to be the *compactness loss* l_c, will be minimized. This loss l_c is computed exclusively considering objects from the target dataset, that are pictures with people inside.

The requirement of descriptiveness is properly evaluated only if there are other elements belonging to different classes with respect to the target class. The one-class dataset is not enough to define a targeted function. As a consequence, an *external dataset* with several classes is introduced in Deep One-class Classification as the *reference dataset*. Objects inside it are completely unlinked with elements of target class.
In order to produce descriptive features, our model shall behave in a good way even in an unrelated classification problem, formulated by the reference dataset. We keep therefore the *cross-entropy loss* to state the descriptiveness of features. This quantity is referred to be the *descriptiveness loss* l_d, that is minimized in order to have high accuracy in classification. This loss l_d is evaluated considering instances only coming from the reference dataset.

It is possible to present a new optimization problem, that considers the above defined loss functions:

$$\hat{g} = \min_{g} l_d(r) + \lambda l_c(t).$$

We emphasize that the two loss functions are computed detached from one another: the descriptiveness loss is based on the external reference dataset r, while the compactness loss relies on the target one-class dataset t, like shown in figure 4.4.
The weighted sum of l_d and l_c defines the *composite loss* l, controlled by the parameter λ.
Minimizing l, features are descriptive and, also, the target class presents a small value of intra-class variance.

4.5 Proposed training architecture

The architecture suggested by Perera and Patel is composed of two identical Convolutional Neural Networks, called *reference network* R and *secondary network* S, that share their weights and work concurrently.

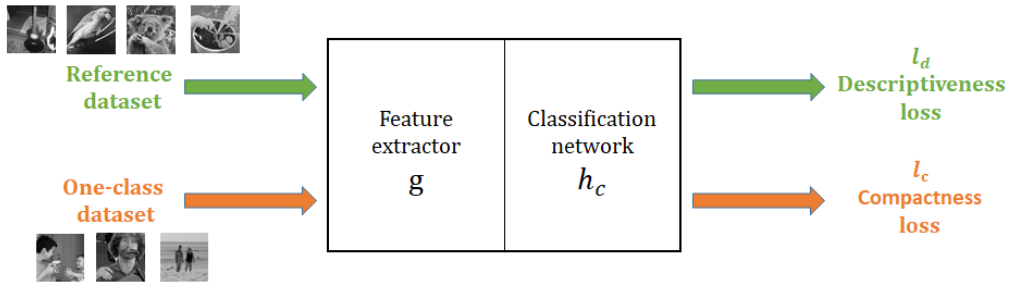


FIGURE 4.4: Descriptiveness and compactness losses

The first network is fed by batches of alien objects from the reference dataset r and produces the descriptiveness loss, while the second one by batches of person images from the target dataset, providing the compactness loss.
The composite loss is then minimized and all weights are updating through the *backpropagation* process.
Since the two networks have the same structure and are characterized by *parameter sharing*, all weights and biases are forced to be equal during the whole training. The proposed training framework is presented in figure 4.5.
In the article, the *AlexNet* network is used with pre-trained weights. The model has all parts frozen up to the last four layers, that produce specialized high-level features and have to adapt to the new task.

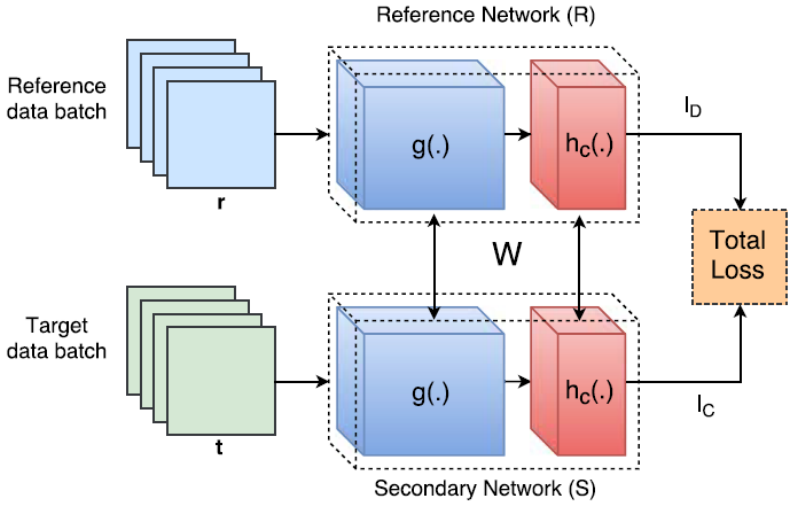


FIGURE 4.5: Proposed training framework. *Source: Perera and Patel* [28]

Chapter 5

Deep One-class Classification of people

In this chapter, the current implementation of the algorithm based on Deep One-class Classification is explained.
First of all, we introduce the *work platform* necessary to develop the model during all phases. Then, the strategy used to create an effective *training dataset* is presented, with the related steps of image processing.
Later, cores of DOC are analysed: the employed network *MobileNetV2*, training and testing frameworks, detailed computations for *loss functions*.
Finally, an overview on *testing datasets* and on *performance measures* used to evaluate all models are shown.

5.1 Work platform

The development of Deep Learning algorithms requires machines with high computational power in order to correcltly handle trainings of complex deep models and large datasets, composed by thousands of images. Using simply a CPU available in common laptop is not enough in this situations, a huge amount of time is needed.
The introduction of GPUs, *Graphics Processing Units*, makes the work easier: they are devices specialized in computer graphics and image processing, able to manipulate in parallel big blocks of data.
The need to have an available GPU has led to employ *Google Colab* [30] and *Jupyter Notebooks* [31].

5.1.1 Google Colab

Google Colab offers the possibility to execute *python* code over the browser, requiring only a Google account and without making any extra setup on your machine. It exploits *Jupyter Notebooks*, that will be clear later, that are saved in Google Drive, with which it easily interfaces.
Google Colab gives free acces to GPUs, that can be Nvidia K80s, T4s, P4s and P100s, depending on the availability. Therefore, there are usage limits that vary over time in order to guarantee resources for free to all.
Since the maximum lifetime of a notebook is 12 hours and, also, long-running computations were stopped after idle periods, the platform is changed.

5.1.2 Jupyter Notebook

Jupyter Notebook is an interactive web application that gives us the possibility to manipulate documents with code, images, text comments and equations.
It is organized in cells that can be executed in chunks; all documents are visualized through a browser page and can be easily shared.
Jupyter Notebooks are run on a computer with *Ubuntu 18.04.5* operating system, that is supplied by two GPUs. The GPU used in the developmet of the algorithm is the *NVIDIA GeForce RTX 2080* with 8GB of memory.

5.1.3 Python libraries

The programming language used to write the code is *Python*.
There are many other options like *R*, very popular in manipulating data for statistic problems, *Java*, a good choice for debugging ease and simplicity with large projects, and *MATLAB*, that offers a framework named Deep Learning Toolbox.
Python is selected because is simple, versatile and presents dedicated frameworks and libraries for Machine Learning.
Libraries used in the thesis project are the following:

- *NumPy:* useful for fundamental computations, including array and matrix manipulations. It contains all popular mathematical function implementations. Tensors are handled by TensorFlow thanks to this library;
- *Scikit-learn:* the most used library in Machine Learning scenario, it includes supervised and unsupervised ML algorithms. It effectively helps in extracting and analysing data. It exploits NumPy and SciPy libraries;
- *Matplotlib:* not closely related to Machine Learning fied, is useful in visualizing data through graphs and plots;
- *OpenCV:* library fundamental for computer vision applications, it contains image processing algorithms that allow to read and write pictures, modifying their colors and dimensions in the needed way.
- *TensorFlow:* most used library for Deep Learning, it is developed by the Google Brain team. It allows to perform complex operations with tensors to train and test deep models in an efficient way.
- *Keras:* is a high-level API used to build deep Neural Networks. In this project, it runs on top of TensorFlow, that makes low-level computations and is referred to be *Keras backend*. Other backends are *CNTK* and *Theano*.

5.2 Training datasets

In this section we explore the two training datasets that characterize Deep One-class Classification: the *target dataset* and the *reference dataset*.

Choosing a right and effective dataset is essential to achieve good results. Everything the network learns is determined by the samples given during the training: the selection of images that make up both datasets directs the evolution of model parameters.

5.2.1 Target dataset

The *target dataset* is composed of instances of the class we want to recognize among all other classes, i.e. of the *person class*. We want a dataset as heterogeneous as possible, that includes most of features that can be later isolated by the CNN as *person features*.
We look for these images in the *Open Images Dataset V4* [32].

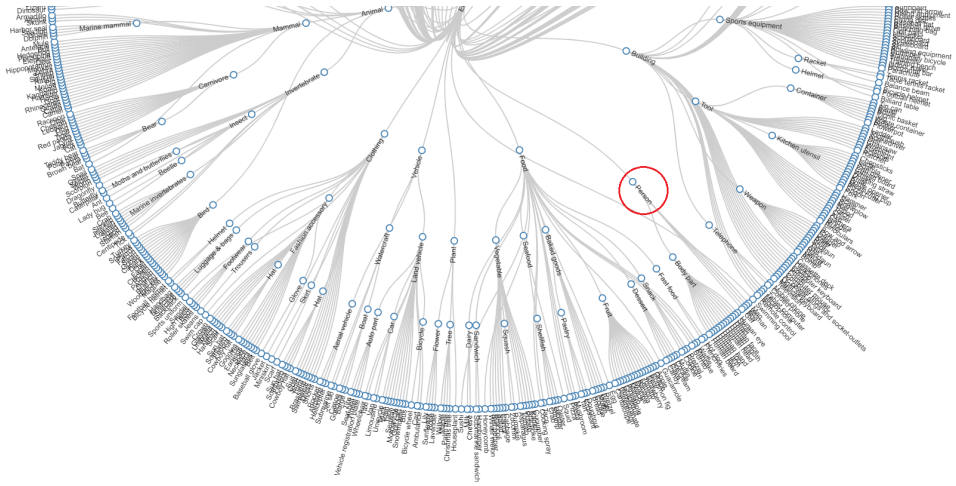


FIGURE 5.1: Bottom part of the dendogram of OID classes where *Person* class is present. *Source:* [32]

Open Images is a collection of about 9 millions of images with image-level labels and object bounding boxes. It is the largest existing dataset with object location annotations, since it includes 14.6M bounding boxes from 600 categories on 1.74M images.
The composition of the dataset for *image classification* we care about, without bounding boxes, is shown in the table 5.1.
The dataset is divided into train, validation and test sets, that span over 19,995 classes. The *Person* class is highlighted in figure 5.1, where bottom part of the dendogram of classes is present.

	Train	**Validation**	**Test**	**NClasses**
Images	9,011,219	41,620	125,436	-
Machine-generated labels	78,977,695	512,093	1,545,835	7,870
Human-verified labels	27,894,289	551,390	1,667,399	19,794

TABLE 5.1: Dataset for image classification in OIDV4. *Source: V. Mazzia and A. Tartaglia* [33]

With the support of the *OIDv4 ToolKit* [33], images of the person class are downloaded: we collected 1761 examples from the *train* set and 9967 examples

from the *validation set*, both with human-verified labels.

Then, all images are meticulously selected. Troubles lie in reaching different types of people, from kids to elders (fig. 5.2), with different gender, skin colors and most disparate physical traits (fig. 5.3). We include also people captured from the front, from the back and from the side (fig. 5.4).
In order to pretty generalize the person properties, we seek for individuals far from the camera, as well as in the foreground.
A particular issue is to understand to what extent parts of human body can be considered still people from the point of view of the algorithm. Are legs, faces, hands, upper bodies instances of people yet? The answer is fairly subjective and depends on our main goal. Examples of images figuring parts of bodies included in the target dataset are shown in picture 5.5.
We decide to use a common criteria in the sorting, in particular, pictures that are removed have:

- people very far from the camera, so described by a too little portion of pixels;
- people not recognizable for various reasons, for example, they are covered by other objects;
- people or parts of bodies situated on the edge of the frame, beacuse they will not appear after the pre-processing part.

(A) (B) (C)

FIGURE 5.2: People of different ages included in the *target dataset*. *Source:* [32]

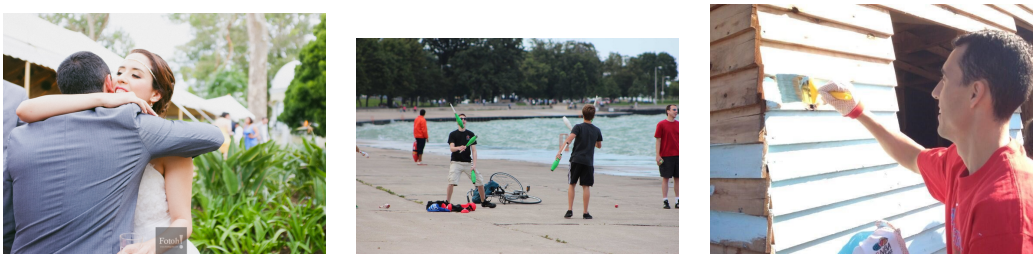
5.2.2 Reference dataset

In Deep One-class Classification aimed at recognizing person instances, it is not possible to collect all alien objects opposed to person class objects. Despite this, we can use an external dataset called *reference dataset* to extract features with the property of *descriptiveness*.
This dataset can be an arbitrary set containing completely unrelated images from multiple classes or, alternatively, a subset of it.

(A) (B) (C)

(D) (E) (F)

FIGURE 5.3: People with different gender, skin colors and physical traits included in the *target dataset*. *Source:* [32]



(A) (B) (C)

FIGURE 5.4: People with different poses included in the *target dataset*. *Source:* [32]

(A) (B) (C)

FIGURE 5.5: Parts of human body included in the *target dataset*. *Source:* [32]

In this work, a subset of the training set of *ILSVR2012* is used.

ILSVRC 2012 is an acronym for *Imagenet Large Scale Visual Recognition Challenge 2012*, a competition of image classification which relies on a subset of the enormous *ImageNet dataset* [34]. The latter contains more than 10 millions of hand-labeled images organized in more than 10 mile categories.
ILSVRC 2012 is organized according to the *WordNet hierarchy* that is based on *synsets*: multiple words or phrases that describe a meaningful concept. Unlike ImageNet, it contains images from just 1000 classes.
The composition of the dataset is presented in the following table 5.2.

	Train	**Validation**	**Test**
Images	1,281,167	50,000	100,000
Size	$\sim$ 156 GB	$\sim$ 7GB	$\sim$ 13GB

Table 5.2: Composition of ILSVRC 2012

The choice of using a subset of the ILSVRC 2012 training set is dictated by memory usage limits and also by the amount of time required in the selection phase.
Big problems arise in the elimination of all the images that contain instances of the target class: we don't want to create interferences between datasets, therefore there must be no people in the pictures of the reference one. The presence of humans is usually very common, that's why each single image is examined and removed if people are present inside.
The reference dataset is composed of randomly chosen 20 classes, each containing 500 images, for a total of 10 thousand of images. In table 5.3, details about classes of the reference dataset are summarized, while in figure 5.6 some images belonging to them are shown.

5.2.3 Image pre-processing

Both defined datasets are composed by color images, that are also called *RGB images* because their pixels are represented by three values, one for each of red, green and blue channels, like explained in section 2.4. However, the model training is carried out using *grayscale images*.
It is a very special choice, so it's time to explain the reasons for this.

One future goal of this discussion, slightly further away, is to extend the Deep One-class Classification in *InfraRed images*, in order to recognize people in frames coming from surveillance videos, even at night.
These videos are taken with special cameras that, instead of visible light, are sensitive to radiations with wavelenghts from $700nm$ to $1mm$. In particular wavelengths in the $700nm$ - $1\mu m$ range characterize the Near-Infrared (NIR) spectrum.
Problems in training networks directly with InfraRed examples arise because there are no large datasets made of enough images like *ImageNet dataset* or *Open Images Dataset*. That's why a training with grayscale pictures is proposed and, then, a generalization to IR datasets through a further transfer learning is suggested.

Class	Synset	Readable name
1	n01443537	goldfish, Carassius auratus
2	n01484850	great white shark, man-eating shark, Carcharodon carcharias
3	n01532829	house finch, linnet, Carpodacus mexicanus
4	n01882714	koala, kangaroo bear, native bear, Phascolarctos cinereus
5	n02128925	jaguar, panther, Panthera onca, Felis onca
6	n02165456	ladybug, ladybeetle, lady beetle, ladybird, ladybird beetle
7	n02279972	monarch butterfly, milkweed butterfly, Danaus plexippus
8	n02494079	squirrel monkey, Saimiri sciureus
9	n02504458	African elephant, Loxodonta africana
10	n02676566	acoustic guitar
11	n02708093	analog clock
12	n02769748	backpack, knapsack, packsack, rucksack, haversack
13	n02823750	beer glass
14	n02948072	candle, taper, wax light
15	n03788195	mosque
16	n03895866	passenger car, coach, carriage
17	n03950228	pitcher, ewer
18	n03991062	pot, flowerpot
19	n04099969	rocking chair, rocker
20	n04264628	space bar

TABLE 5.3: Classes of the *reference dataset*

Later, some test datasets will be presented, including also an InfraRed one that is tested using resulting models.

Valid graysacale reference and target datasets are created by the corresponding RGB datasets, selected as shown before, maintaining the same vastness and heterogeneity.
Moreover, pictures have a wide range of sizes, so they must be transformed to have all equal elements in the collection. The library *OpenCV* is used to process all images.
The following steps are carried out:

- each image is centrally cropped along its smaller size. In this way we can resize it without altering the image aspect ratio and the properties of objects within. Examples of wrong resizing that produces distorted examples are shown in figure 5.7;
- each picture is resized to square format of 224×224 with a bilinear interpolation. Usually, it is the default input shape in networks and it is kept also in this work because handle many (224, 224, 3) images is not so expensive in terms of memory usage: our dataset composed by 16,000 resized images occupies about 371 MB, against the previous 2.7 GB;
- each image is made a grayscale image with size of (224, 224, 1), having a single channel;
- each grayscale image is brought back on three channels, through the *merge* command. The single channel is repeated three times, providing images

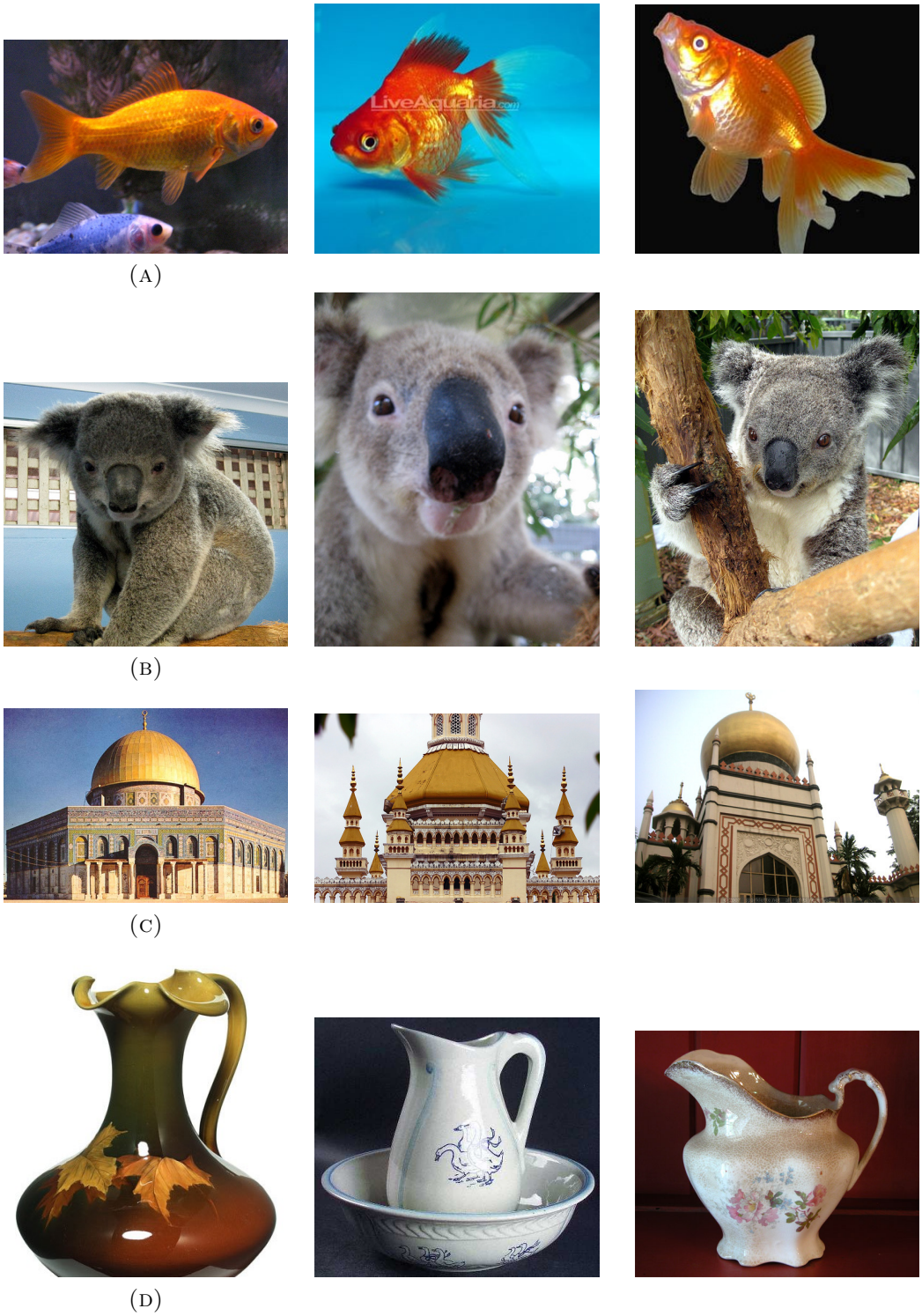


FIGURE 5.6: Images incuded in the *reference dataset* with synsets: (A) *n01443537* (B) *n01882714* (C) *n03788195* (D) *n03950228*. *Source:* [34]

of size (224, 224, 3) again. This operation is done since the structure of most of networks presents a three channel configuration.

(A)

(B)

(C)

FIGURE 5.7: Per line, on the left: *original image*, in the middle: *distorted image*, on the right: *correctly pre-processed image*

5.3 Deep One-class Classification cores

In this section, the cores of the method are presented: the employed network and the computed losses in our Deep One-class Classification.
We used a different kind of model than the suggested nets in the article, AlexNet and VGG16, that is *MobileNetV2*.

5.3.1 DOC backbone: MobileNetV2

MobileNetV2 by Google belongs to MobileNets family, efficient and optimized architectures for mobile devices and, in general, all those which have poor resources.
It is our choice because it is fast and provides high accuracy, requiring few parameters and low computational power, also compared to previous versions.

This model is a refinement of MobileNetV1, that contains repetitions of *depthwise separable convolutions* instead of the expensive and slower normal convolutions. The main building block in MobileNetV1 is the following (fig. 5.8):

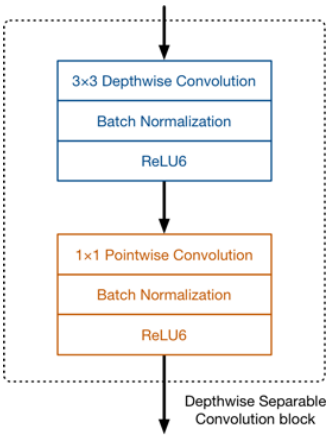


FIGURE 5.8: Building block of MobileNetV1. *Source: M. Hollemans* [35]

This is the depthwise separable convolution block: the input passes through a 3×3 depthwise convolutional layer, that makes a lightweight filtering, and, then, through a 1×1 pointwise convolutional one, that creates new features. This operations are followed by a batch normalization and by an activation function that is *ReLU6*: $min(max(x, 0), 6)$.

In MobileNetV2 there are two changes: *linear bottlenecks* and *residual connections*, that facilitate the flow of gradients in parameter updating.
The main building block in MobileNetV2 is shown in figure 5.9: it is a bottleneck depthseparable convolution block with residuals.

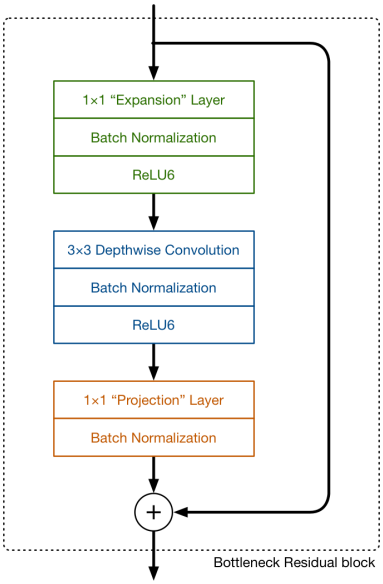


FIGURE 5.9: Building block of MobileNetV2. *Source: M. Hollemans* [35]

Here, three operations are performend:

- a 1×1 convolution thanks to the expansion layer, that increases the number of channels before data goes into next layer, basing on the hyperparameter *expansion factor* whose default value is 6;
- a 3×3 depthwise convolution;
- a final 1×1 convolution by the projection layer, that reduces the number of channels unlike before, where it is kept the same or doubled. This last layer is actually called *bottleneck layer* since it decreases the quantity of data flowing through the network and behaves like a bottleneck.

Also, blocks have a batch normalization layer and the activation function ReLU6, that is missing in the projection layer because it provides low-dimensional output that can be altered using a non-linearity.

The entire architecture of MobileNetV2 is made of 17 identical building blocks (only the first one has a traditional 3×3 convolution with 32 channels rather than the expansion layer) and, then, there are in sequence: a 1×1 convolutional block, a global average pooling layer and, finally, a classification layer.
This model works so well because tensors remain relatively small flowing through the net, reducing the number of computations, but, at the same time, are expanded when filtered to extract right features.
Both expansion and projection layers act with learnable parameters, so they learn how to best decompress and compress data step by step, during the whole training.
In figure 5.10, the block of MobileNetV2 provided by *Keras API* is shown, consistent with the one in picture 5.9.

Layer (type)	Output Shape	Param #
block_1_expand (Conv2D)	(None, 112, 112, 96)	1536
block_1_expand_BN (BatchNormali	(None, 112, 112, 96)	384
block_1_expand_relu (ReLU)	(None, 112, 112, 96)	0
block_1_pad (ZeroPadding2D)	(None, 113, 113, 96)	0
block_1_depthwise (DepthwiseCon	(None, 56, 56, 96)	864
block_1_depthwise_BN (BatchNorm	(None, 56, 56, 96)	384
block_1_depthwise_relu (ReLU)	(None, 56, 56, 96)	0
block_1_project (Conv2D)	(None, 56, 56, 24)	2304
block_1_project_BN (BatchNormal	(None, 56, 56, 24)	96

FIGURE 5.10: Building block of MobileNetV2 in Keras API

The architecture MobileNetV2 is imported for our purpose, loading weights pre-trained on ImageNet and without including the default top part with 1000 neurons.
The input shape of images we provide is set to (224, 224, 3), in accordance with 224×224 size and 3 channels in the pre-processing part.

The hyperparameter *alpha*$\in (0, 1]$, known as the *width multiplier* that determines the number of filters at each layer, is set to its default value 1.
A *global average pooling layer* is inserted after the the last convolutional block, passing from a 4D output tensor of shape (batch_size, 7, 7, 1280) to a flattened 2D output tensor of shape (batch_size, 1280). These 1280 numbers are the features extracted from the input images, from which we minimize the compactness loss and on which we base the classification of objects.
Finally, a *fully connected layer* with a *softmax* activation function and with a number of units equal to the total classes of the reference dataset (20) is attached, in order to compute the descriptiveness loss.

5.3.2 Training architecture

In our solution a unique MobileNetV2 is instantiated, in contrast to what is suggested in the article (explained in section 4.5). Having two identical networks, that run in parallel and share weights, requires the double amout of memory: the learning process is very hard to handle.
New ways to train this unique network are proposed below.

The network is fed with a big input batch, that is composed of two smaller batches of the same size, called mini-batches: they are the *target sub-batch* and the *reference sub-batch*.
As the names suggest, the first one is composed by simply images of people from the target dataset, while the second one by images coming only from the reference dataset.
We use a function with multiple *Keras ImageDataGenerator* objects to handle this kind of training, shown in listing 5.1.

Listing 5.1: Code for the generation of mixed input batches

```
input_imgen = ImageDataGenerator(preprocessing_function = preprocess_input)

def generate_generator_multiple(generator, dir1, dir2, sub_batch_size,
                                img_height, img_width, n_classes):

    genX1 = generator.flow_from_directory(dir1,
                                          target_size = (img_height, img_width),
                                          class_mode = 'categorical',
                                          batch_size = sub_batch_size,
                                          shuffle=True)

    genX2 = generator.flow_from_directory(dir2,
                                          target_size = (img_height,img_width),
                                          class_mode = 'categorical',
                                          batch_size = sub_batch_size,
                                          shuffle=True)
    while True:
            X1i = genX1.next()
            X2i = genX2.next()
            yield np.concatenate([X1i[0], X2i[0]]),
                  np.concatenate([to_categorical(np.argmax(X1i[1], axis=1),
                                  num_classes=n_classes_ref), X2i[1]])

inputgenerator = generate_generator_multiple(generator = input_imgen,
                                             dir1 = path_target,
                                             dir2 = path_reference,
                                             sub_batch_size = sub_batch_size,
                                             img_height = 224,
                                             img_width = 224,
                                             n_classes = n_classes_ref)
```

In this way all images are provided to the network "on the fly", without storing all matrices in memory and causing related memory issues.

From the target sub-batch, having within just pictures of individuals, we are interested in features that characterize the target class. The meaningful output coming from this mini-batch is the one produced by the *average pooling layer*: 1280 values for each image of the batch, that are the person features we want to impose as close as possible in the compactness loss computation.
From the reference sub-batch, instead, we are interested in the classification output provided by the *fully connected layer*: 20 values that rapresent the categorical label of one among the 20 classes from the reference dataset. The cross-entropy loss is evaluated to impose descriptiveness in features.
Therefore, the network has two parallel outputs from which the two losses are computed:

- the global average pooling layer output, representing the extracted features from MobileNetV2;
- the fully connected layer output, representing the classification predictions.

5.3.3 Training settings

Before discussing about loss computations, an overview of training settings is present.

- In the training phase some layers of the network are *frozen*, to preserve imported parameters pre-trained on ImageNet. This means we take low-level features learned in a different classification task, by leveraging them in our problem.
  In the article, Perera and Patel decide to leave unfrozen the last four layers of the AlexNet architecture, that are *Conv5*, *FC6*, *FC7* and *FC8*. However, this model is very simple since it contains a total of 8 layers.
  In our MobileNetV2 network we think differently, in terms of blocks. We choose initially to freeze all blocks until *block 13*, having 40 unfrozen layers over the whole 157 layers.
  Further analysis are done in the next chapter, understanding how the number of trainable layers affects the model performance;
- Wa start using a batch size of 256 samples, resulting in a sub-batch size equal to 128.
  In the next chapter, additional tests are performed by varying the batch size;
- The value of lambda λ is a crucial point in the discussion, because it controls the mutual importance between the two terms in the composite loss defined below, where r is the reference dataset and t is the target one:

  $$l(r,t) = l_d(r) + \lambda l_c(t).$$

  It is initially set equal to 0.1, as suggested by Perera and Patel, giving more prominence to the descriptiveness loss. Actually, the orders of magnitude

of l_d and l_c in the article differ from ours, so λ is changed to produce a more accurate minimization.
In the next chapter, the best value for lambda is found, basing on the trend of metrics ROC AUC, F1 score, precision and recall;

- Epochs are delineated from the size of the target dataset.
  The number of epochs in the simulation is is set to 400, taking care to save intermediate models every 50 epochs to properly study the evolution of tested metrics;

- The optimizer is the *gradient descent* algorithm, employed with a very low learning rate $l_r = 0.00005$ and a weight decay of 0.00005.

5.3.4 Compactness loss computation

The *compactness loss* is described by the custom function in listing 5.2.

Listing 5.2: Code for compactness loss computation

```
def compactness_loss(y_true, y_pred):

    y_pred_target = y_pred[0:sub_batch_size]

    l_c = tf.keras.backend.mean(tf.keras.backend.var(y_pred_target,
                                axis = 0, keepdims=False))

    return l_c * beta
```

Therefore, there are two input quantities:

- *y_true*: the true labels of the batch, of size (batch_size, n_classes_ref). This quantity is not used in the l_c computation because it has no role in imposing similarity among person features;

- *y_pred*: predictions of the intermediate features for each element in the batch, of size (batch_size, n_features).
  It is produced by the average pooling layer, so the number of features is 1280. We choose this layer because it has weights pre-trained on ImageNet, that speed up the learning process compared to those with random inizialization.

In order to consider only features of person images, the first half part of the batch is isolated.
Then, the following operations are performed: the variance of the feature distribution along the batch for each feature and the mean of all variances.
This number is multiplied by a correction factor *beta* β that is:

$$\beta = sub\_batch\_size^2/(sub\_batch\_size - 1)^2.$$

Minimizing the mean of the variance of all the features implies having similar characteristics for all images representing people.

5.3.5 Descriptiveness loss computation

The *descriptiveness loss* is described by the custom function in listing 5.3.

LISTING 5.3: Code for descriptiveness loss computation

```
cce = tf.keras.losses.CategoricalCrossentropy(from_logits=False)

def descriptiveness_loss(y\_true, y\_pred):

    y_true_reference = y_true[sub_batch_size:batch_size]
    y_pred_reference = y_pred[sub_batch_size:batch_size]

    l_d = cce(y_true_reference, y_pred_reference)

    return l_d
```

Also here, there are two input quantities:

- *y_true*: the true labels of the batch, of size (batch_size, n_classes_ref). This quantity is provided by the *inputgenerator* in 5.1;
- *y_pred*: predictions coming from the last fully connected layer, of size (batch_size, n_classes_ref). The second dimension n_classes_ref is 20, corresponding to the categorical label of classes from the reference dataset. The label of the person class is not included because this is not a multi-class classification problem: we focus on person features, not on discriminating label mismatches, since there could be alien objects belonging to unknow categories.

The descriptiveness loss is computed with respect to only elements of the reference dataset. Therefore, the second half part of the batch is considered both in *y_true* and in *y_pred*.
The first part of them contains meaningless numbers, because we don't care about person image labels.
Then, the *categorical cross-entropy loss* is evaluated between the predicted labels and the desired ones and it is minimized to realize a good classification.
In this way, features are characterized by the property of descriptiveness, in addition to compactness.

5.3.6 Testing framework

The testing part is performed though an appropriate framework shown in figure 5.11.
The block called *g(·)* is a piece of the trained MobileNetV2 that produces the 1280 numbers corresponding to the features, from the first block to the global average pooling layer. In the code it is called *model_features*.
Two steps are carried out to determine if a picture contains people: *template generation* (upper part) and *matching*, realized by the classifier.

In *template generation* phase, some samples $\mathbf{v}=\{v_1, v_2, ...v_n\}$ are selected from the target dataset and are sent into the network g: the resulting quantity $g(\mathbf{v})$

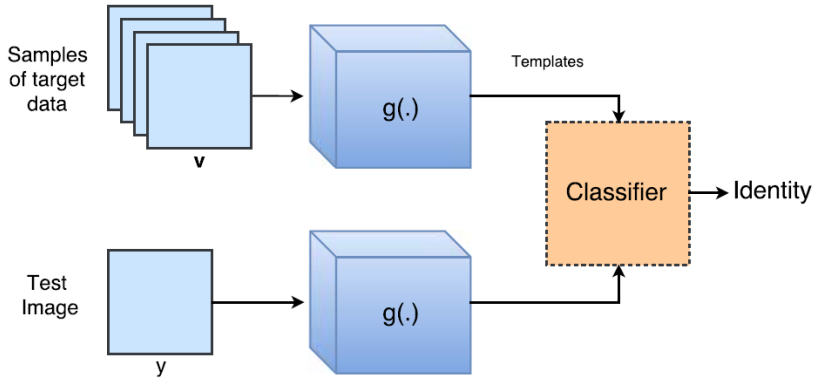


FIGURE 5.11: Testing framework. *Source: Perera and Patel* [28]

stores the so-called *templates*, the features extracted by each image v_i with people inside, belonging to a target sub-set.
In *matching* phase, we firstly extract thanks to g the features related to the test image y. Then, features $g(y)$ are evaluated through a matching function f, which compares them to templates $g(\mathbf{v})$ and produces a *score*.
In listing 5.4, the matching procedure that generates scores from a testing dataset of n_test images is shown.
In the implementation, the matching function is the *Euclidean distance*.

LISTING 5.4: Code for matching phase

```
def scores_generation(features_test, templates):

    for f in features_test:

        d = [np.linalg.norm(f-t) for t in templates]
        distances_vector.append(d)

    scores = np.amin(distances_vector, axis=1)
    scores = np.array(scores)

return scores
```

The provided inputs are:

- *features_test*: features extracted from test images, of size (n_test, n_features);
- *templates*: stored templates corresponding to baseline characteristics of the person class, of size (n_templates, n_features).

Features coming from each test image are compared to all templates: the quantity d contains the euclidean distances between them, sizing (n_templates,).
Each vector d is computed for all images in the test dataset and is saved in the *distances_vector*, of size (n_test, n_templates).
The scores, stored in the vector *scores* of size (n_test,), are selected taking the

minimum value among all computed distances in *distances_vector*, for all images.

Finally, scores are transformed in considerable output for One-Class Classification thanks to a threshold δ:

$$output[i] = \begin{cases} 0: & scores[i] \leq \delta \\ 1: & scores[i] > \delta \end{cases}$$

considering the label 0 as an object belonging to the target class *person* and the label 1 to the alien class *others*.

The appropriate value for δ depends on the application.
The chosen δ in our Deep One-class Classification is the one that maximizes the quantity (TNR-FNR), producing an high value of TNR, the True Negative Rate, and a low value of FNR, the False Negative Rate.
The first one indicates the ratio of negative instances correctly classified as negative, while the second one is the ratio of positive instance incorrectly classified.
Therefore, maximizing the term (TNR-FNR) allows to reach an high value of intances classified as people that are actually people and a low value of alien objects wrongly classified as people.
Considering also that TNR=1-FPR and FNR=1-TPR, finding the maximum value for (TNR-FNR) means maximizing (TPR-FPR) (True Positive Rate - False Positive Rate), that corresponds to the closest point to the top left corner of the ROC curve.
More details about these and other metrics are presented in next sections.

The number of templates is set to 40, as suggested in the article.
Further analysis are done to understand the impact of the number of them.

5.4 Testing datasets

In order to evaluate the performance of models in Deep One-class Classification, three different testing datasets are employed.

5.4.1 Dataset 1: *BN1*

The first testing dataset is composed of 2000 *grayscale images*: 1000 with people and 1000 without individuals, but including alien objects from the 20 classes of the reference dataset.
All the pictures are fresh examples from the point of view of the algorithm.
The composition of BN1 is presented in table 5.4.

5.4.2 Dataset 2: *BN2*

The second testing dataset is composed of 2000 *grayscale images*: 1000 with people and 1000 including alien objects from different classes of the reference

N. of images	Synset	Readable name
50	n01443537	goldfish, Carassius auratus
50	n01484850	great white shark, man-eating shark, Carcharodon carcharias
50	n01532829	house finch, linnet, Carpodacus mexicanus
50	n01882714	koala, kangaroo bear, native bear, Phascolarctos cinereus
50	n02128925	jaguar, panther, Panthera onca, Felis onca
50	n02165456	ladybug, ladybeetle, lady beetle, ladybird, ladybird beetle
50	n02279972	monarch butterfly, milkweed butterfly, Danaus plexippus
50	n02494079	squirrel monkey, Saimiri sciureus
50	n02504458	African elephant, Loxodonta africana
50	n02676566	acoustic guitar
50	n02708093	analog clock
50	n02769748	backpack, knapsack, packsack, rucksack, haversack
50	n02823750	beer glass
50	n02948072	candle, taper, wax light
50	n03788195	mosque
50	n03895866	passenger car, coach, carriage
50	n03950228	pitcher, ewer
50	n03991062	pot, flowerpot
50	n04099969	rocking chair, rocker
50	n04264628	space bar

TABLE 5.4: Composition of testing dataset *BN1*

dataset. In BN2, images of the negative category belong to 25 external classes, like shown in table 5.5.

5.4.3 Dataset 3: *IR*

The third testing dataset is smaller than previous ones because it contains the few *InfraRed images* found on web for our purpose: 55 images with people and 55 images with alien objects different from individuals.
Unlike the others, the number of templates here are 20, separately selected because there is no a target dataset with InfraRed images.
Examples of images belonging to IR dataset are shown in figure 6.10.
Sources of IR testing dataset are: [36], where [37] [38] are available, [39] , [40], [41], [42], [43] and [44].

5.5 Performance measures

All obtained models for *person recognition* are compared through some metrics and graphical tools.
Note that Deep One-class Classification provides a binary outcome: the value *0* represents target or normal images including people, while the value *1* means that only alien objects different from person intances are present in the picture. Therefore, performances are evaluated using the same metrics of binary classification.
In this section we explain what are these measures, where they come from and why they are so effective in evaluating model performances.

Class	**N. of images**	**Synset**	**Readable name**
1	40	n01616318	vulture
2	40	n03637318	lampshade, lamp shade
3	40	n03857828	oscilloscope, scope, cathode-ray oscilloscope, CRO
4	40	n03982430	pool table, billiard table, snooker table
5	40	n04019541	puck, hockey puck
6	40	n04040759	radiator
7	40	n04041544	radio, wireless
8	40	n04070727	refrigerator, icebox
9	40	n04074963	remote control, remote
10	40	n04209239	shower curtain
11	40	n04228054	ski
12	40	n04265275	space heater
13	40	n04317175	stethoscope
14	40	n04355338	sundial
15	40	n04380533	table lamp
16	40	n04428191	thresher, thrasher, threshing machine
17	40	n04493381	tub, vat
18	40	n04501370	turnstile
19	40	n04507155	umbrella
20	40	n04525305	vending machine
21	40	n04579145	whiskey jug
22	40	n07584110	consomme
23	40	n07716906	spaghetti squash
24	40	n07754684	jackfruit, jak, jack
25	40	n12768682	buckeye, horse chestnut, conker

TABLE 5.5: Composition of testing dataset *BN2*

All the metrics below refer to these kinds of predictions:

- *True Positive* (TP) = instances of the positive class correctly classified;
- *False Positive* (FP) = instances of the positive class wrongly classified;
- *True Negative* (TN) = instances of the negative class correctly classified;
- *False Negative* (FN) = instances of the negative class wrongly classified;

A perfectly performed classification would produce zero false positive and false negative predictions (FP=FN=0).

5.5.1 Precision

The *precision* of a classifier is the metric that evaluates the accuracy of positive predictions, defined as:

$$precision = \frac{TP}{TP + FP}$$

It quantifies, across all classified positive examples, how many are actually positive.

(A)

(B)

FIGURE 5.12: Images incuded in the *IR* dataset: (A) *person class* (B) *alien class*

5.5.2 Recall

The *recall* of a classifier is also called *sensitivity* or *true positive rate* (TPR). It measures how many positive instances are correctly identified:

$$recall = \frac{TP}{TP + FN}$$

5.5.3 F1 score

In order to incorporate precision and recall into a unique metric, the *F1 score* is defined as follows:

$$F1 = \frac{2}{\frac{1}{precision} + \frac{1}{recall}} = 2 \times \frac{precision \times recall}{precision + recall}$$

It is the harmonic mean of the two metrics, signifying that an high F1 score is produced only when both precision and recall are similar and high.

Rember that precision and recall cannot reach both their best values: increasing one of them decreases the other one, like explained in figure 5.13.

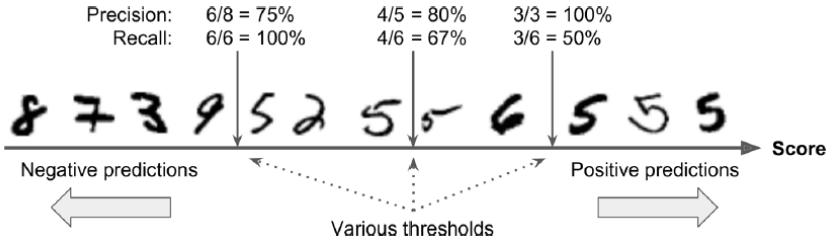


FIGURE 5.13: Precision/Recall trade-off in a "5-detector" (5 is the *positive class* and non-5 is the *negative class*). *Source: A. Géron* [1]

5.5.4 Accuracy

The *accuracy* measures how many examples, among positive and negative ones, are correctly classified. It is defined as:

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

5.5.5 ROC curve

The *Receiver Operating Characteristic* curve plots the *True Positive Rate* (TPR), that is the recall, versus the *False Positive Rate* (FPR) for all possible thresholds (figure 5.14).

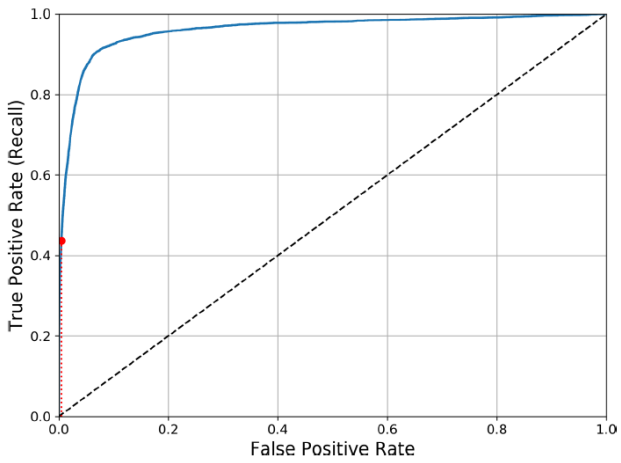


FIGURE 5.14: ROC curve. *Source: A. Géron* [1]

The FPR quantifies how many negative instances are incorrectly classified and is defined as follows:

$$FPR = \frac{FP}{FP + TN} = 1 - TNR$$

The quantity TNR, also known as *specificity*, is the *True Negative Rate* and identifies the ratio of negative instances correctly classified as negative.

In other words, in the ROC we see the evolution of *sensitivity* against (*1-specificity*).

A merely random classifier presents a straight diagonal ROC curve, as the dotted line in figure 5.14, while a good classifier has a ROC curve near the top left corner.

5.5.6 AUC

The *Area Under the Curve* (AUC) of the ROC curve is another metric to evaluate a classifier. A ROC AUC equal to 1 means that a perfect classification is performed, an AUC value of 0.5, instead, refers to a random classifier.

5.5.7 DET curve

The *Detection Error Tradeoff* curve plots the *False Positive Rate* (FPR) against the *False Negative Rate* (FNR) for all possible threshold values (figure 5.15).
It is an alternative way to visually evaluate performances of different classifiers over the ROC curve.
DET curves are more distinguishable in the plot area, differently from ROC curves that tends to be all overlapped in the top left corner.

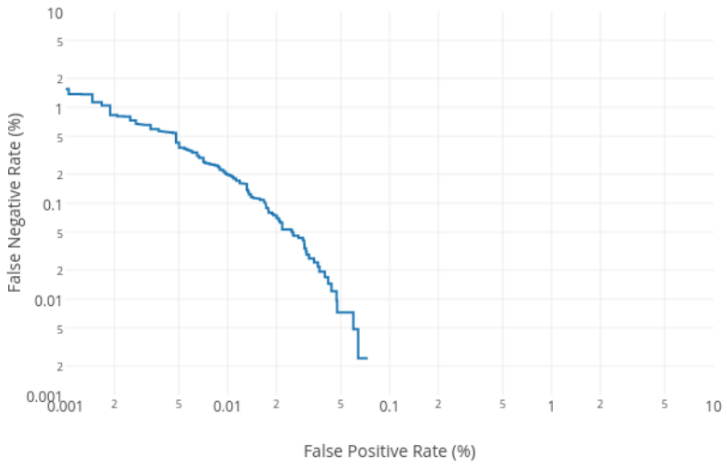


FIGURE 5.15: DET curve. *Source: J. Karnowski* [45]

5.5.8 t-SNE

The *t-distributed Stochastic Neighbor Embedding* (t-SNE) is a probabilistic dimensionality reduction technique for visualizing high-dimensional data in a low-dimensional space. It is the *t-distributed* variant by Laurens van der Maaten of the Stochastic Neighbor Embedding proposed by Sam Roweis and Geoffrey Hinton.
We use this tecnique to visualize features learned by Deep One-class Classification algorithm compared to those produced by the binary classification algorithm. Through t-SNE, each feature represented by 1280 values is modeled by

a two-dimensional point.
The main steps of the algorithm are:

- a probability distribution over couple of high-dimensional points is constructed by t-SNE, assigning a higher probability to similar ones and lower probability to unlike ones;
- after defining a similar probability distribution among objects in the low-dimensional space, t-SNE makes lower the *Kullback–Leibler divergence* between the two distributions basing on point locations.

In this way, objects that are similar in high-dimensional space are modeled by close points in the two-dimesional space, while objects that are dissimilar by distant points.

Figure 5.16 shows clustered digits of MNIST dataset (each described by a 785-dimensional vector) in a 2D space, after dimensionality reduction by t-SNE.

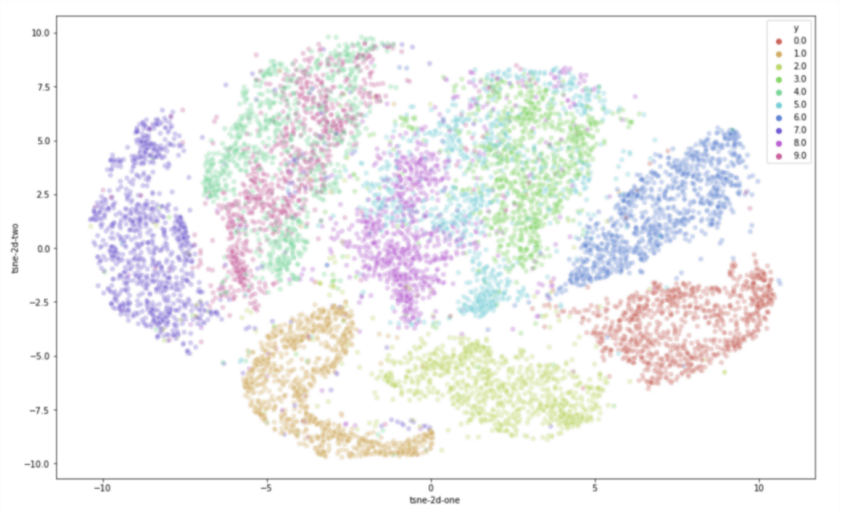


FIGURE 5.16: Visualization of MNIST dataset through t-SNE.
Source: L.Derksen [46]

Chapter 6

Results and conclusions

In this chapter, all obtained results are shown through tables and plots, finally including conclusions and suggestions on future work.

6.1 Results

Detailed analysis are performed to understand the impact of changing some parameters and, later, find the best model for Deep One-class Classification of people.
In particular, we operate:

- modifying λ;
- changing the number of trainable layers;
- varying the input batch size;
- modifying the number of templates;
- decreasing the number of target examples available during the training.

Measures and graphical tools used in evaluating model performances are previously explained in section 5.5. Metrics are precision, recall, F1 score, accuracy and ROC AUC, while graphical tools are ROC curves, DET curves and t-SNE visualization of features.

But first, we must pay attention to the meaning that precision and recall have in our discussion, because these kinds of metrics are closely related to the chosen positive class.
In Deep One-class Classification, however, we represent the person class using the value 0 (negative class) and the alien class using the value 1 (positive class). If we want to refer all metrics to the target class, we need to reverse label values produced by DOC models.
In this way, consistent results are generated through an unbiased comparison among all models, including binary classification ones.
We can say now that *precision* quantifies, across all examples classified as person, how many are actually person and *recall* measures how many instances of people are correctly identified.

6.1.1 Impact of λ

Here, the impact of the main parameter λ is studied.
It is so relevant because it controls the mutual importance between the two losses that realize the composite loss. We need, therefore, to understand which classification results come from if we tip one way or the other.

Table 6.1 shows values of ROC AUC, precision, recall, F1 score and accuracy referred to testing dataset *BN1* (described in section 5.4), using a wide range of lambda. Values are kept after a training of 400 epochs.
We selected 400 as the final epoch, because is when the composite loss converges without being more minimized. To ensure the correctness of this value, all metrics are monitored every 50 epochs, keeping track of their variations.
Other settings in all the performed training phases are: the target dataset is composed of 6000 images, the trainable layers in MobileNetV2 model are 40 and the input batch size is equal to 256.

λ	AUC	Precision	Recall	F1 score	Accuracy
0.1	0.949	0.922	0.845	0.882	0.887
0.5	0.965	0.959	0.859	0.906	0.911
1	0.979	0.949	0.918	0.933	0.935
5	0.992	0.959	0.965	0.962	0.962
10	0.992	0.97	0.958	0.964	0.964
50	0.995	0.963	0.983	0.973	0.974
100	0.994	0.968	0.963	0.965	0.966
400	0.976	0.927	0.929	0.928	0.928

TABLE 6.1: Performance metrics of *BN1* by changing λ

Models with λ=50 and λ=100 presents the best metric values.

In figure 6.1, ROC curves of all models are compared too.
Notice that best models have ROC curves very close to the top left corner of the plot that overlap (light blue with λ=5, red with λ=10, green with λ=50, black with λ=100).
A better visualization of model performances is provided by plotting DET curves (fig. 6.2), where each line is more distinguishable from others.
The curve with λ=50 is placed lower, so it has best values of FPR and FNR.

The trends of composite loss, compactness loss and descriptiveness loss are finally analysed to derive a general rule in choosing λ.
We consider losses in three key scenarios: when λ= 0.1, λ= 400 and λ= 50 (in figure 6.3).
Values of losses are stored every 10 batch iterations, in order to understand what happens during each epoch: since epochs are 400 and batches are 46 (6000 divided by mini-batch size of 128, rounded down to the lower integer), 2000 values are visualized, 5 for each epoch.

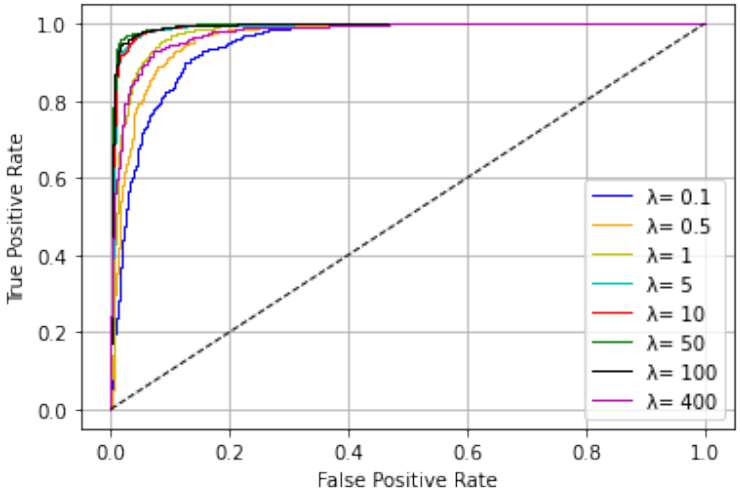


FIGURE 6.1: ROC curves with different λ

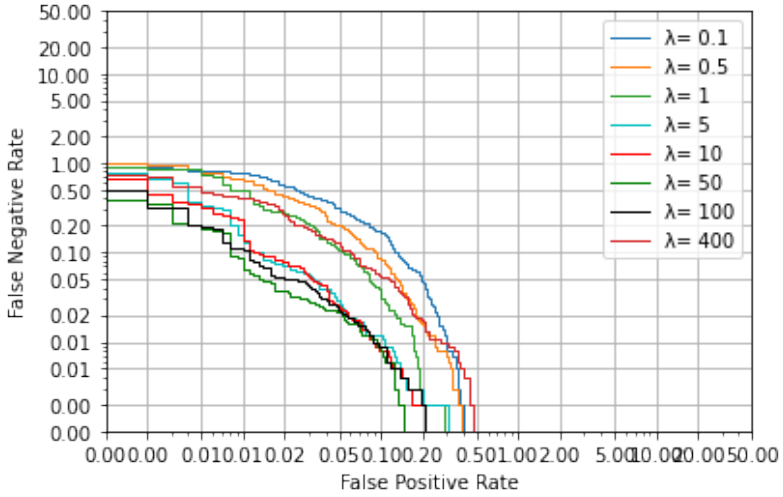


FIGURE 6.2: DET curves with different λ

We can therefore come to some conclusions about what binds λ and the range of losses:

(A) λ= 0.1 $l_c \in [0.33, 0.24]$ $l_d \in [3.5, 0.25]$
In this scenario, multiplying l_c by λ equal to 0.1 produces a composite loss completely dominated by the descriptiveness. It follows that l_d is correctly minimized, while l_c is not enough reduced.
The resulting model is able to categorize in a good way objects, but the variance of extracted features is not very low.

(B) λ= 400 $l_c \in [0.3, 0.0008]$ $l_d \in [3.5, 1.5]$
In this scenario, multiplying l_c by λ equal to 400 produces a composite loss completely dominated by the compactness. It follows that l_c is correctly minimized, while l_d cannot reach low values.
The resulting model is able to isolate features of people, but classification is not

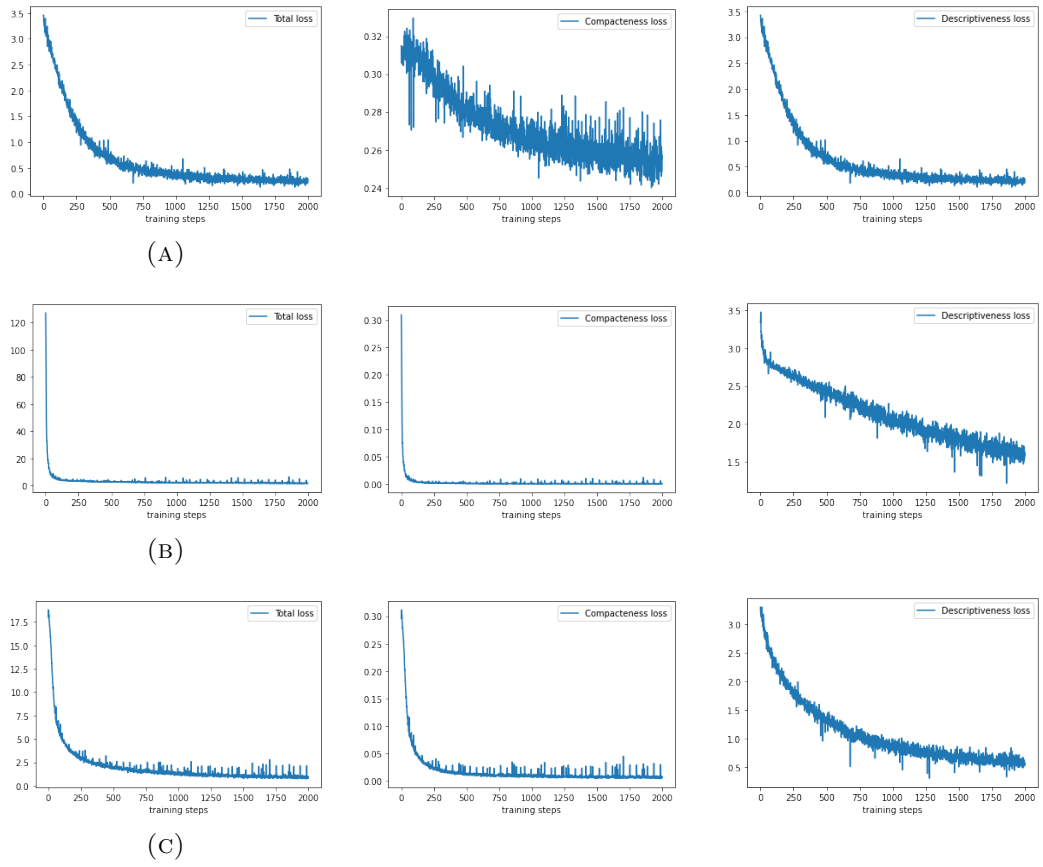


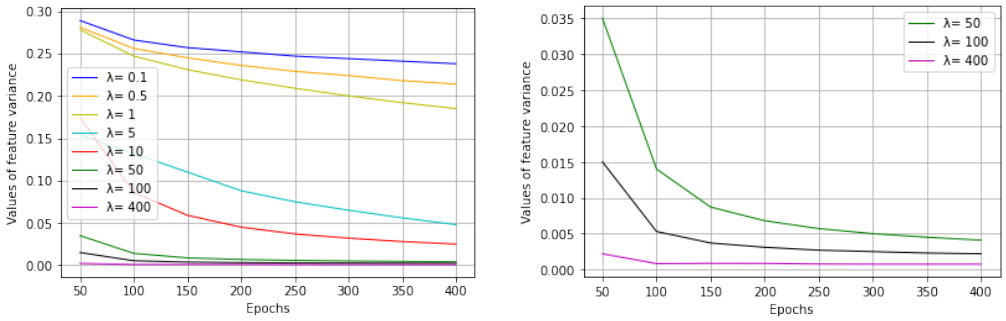
FIGURE 6.3: Composite loss, descriptiveness loss and compactness loss when: (A) λ=0.1 (B) λ=400 (C) λ=50

correctly performed, producing bad metrics.

(C) λ= 50 $l_c \in [0.3, 0.004]$ $l_d \in [3.3, 0.5]$
In this scenario, multiplying l_c by λ equal to 50 produces a composite loss balanced between compactness and descriptiveness. It follows that both losses are correctly minimized.
The resulting model is able to isolate features of people and, at the same time, to categorize in a good way all objects. This is why all performance measures reach high values here.

Additionally, in figure 6.4 is possible to see that the variance of features from people images decreases in all λ cases keeping up with the training, but at different levels.
Cases with λ=50, λ=100, λ=400 (zoomed in fig. 6.4 (B)) have values of variance very very small, but they lose the capability of producing descriptive features.

The best DOC model should produce similar target features through compactness loss minimization and, at the same time, should correctly categorize elements of other classes through descriptiveness loss minimization.
Considering the batch size of 256, we chose the model with λ=50 as the best one, since it has the highest level of AUC, recall, F1 score and accuracy, it is



(A) (B)

FIGURE 6.4: Decreasing of feature variance in all λ cases

placed lower in DET curve plot and presents a correct loss minimization. This best model will be later compared to the related binary model.

6.1.2 Impact of the number of trainable layers in DOC

Here, the impact of number of layers that are learning during the training is studied. Metric values of dataset *BN1* are reported in table 6.2, after a training of 400 epochs using parameters: λ= 50, batch size= 256, elements of target dataset= 6000.

Frozen till	Train. layers	AUC	Precision	Recall	F1 score	Accuracy
block_16	last 13	0.993	0.973	0.967	0.97	0.97
block_15	last 22	0.995	0.979	0.968	0.973	0.974
block_14	last 31	0.994	0.968	0.964	0.966	0.964
block_13	last 40	0.995	0.963	0.983	0.973	0.974
block_12	last 49	0.995	0.964	0.979	0.972	0.972
block_11	last 58	0.995	0.974	0.97	0.972	0.972

TABLE 6.2: Performance metrics of *BN1* by changing the number of trainable layers

Notice that from a certain number of trainable layers (22) onwards, except for case with 31 trainable layers, ROC AUC is stable at 0.995, while F1 score and accuracy present very similar values.
The scenario where there are frozen layers till *block 13* is preferred (setting the parameter *trainable = True* for last 40 layers) because has an higher recall, 0.983, on equal terms.

6.1.3 Impact of the input batch size

We start using an input batch of 256 images, composed of target and reference dataset examples as described in section 5.3.2, because 256 is the largest allowed batch size, without creating memory overflow errors.
In order to evaluate the impact of the batch size in Deep One-class Classification, the number of elements that are fed into the network in parallel is varied,

decreasing it until 32 passing by power of 2 sizes.
In table 6.3, metric values referred to dataset *BN1* at epoch 400 are shown. Other parameters in the performed trainings are: number of target elements = 6000 and number of trainable layers = 40.
The value λ changes from 50 to 10 when decreasing the batch size, because the order of magnitude of losses varies. The parameter λ is therefore adapted to produce a balanced composite loss between compactness loss and descriptiveness loss.

Batch size	λ	AUC	Precision	Recall	F1 score	Accuracy
256=128+128	50	0.995	0.963	0.983	0.973	0.974
128=64+64	10	0.995	0.977	0.97	0.973	0.974
64=32+32	10	0.997	0.985	0.976	0.98	0.981
32=16+16	10	0.997	0.989	0.976	0.982	0.983

TABLE 6.3: Performance metrics of *BN1* by changing the input batch size

It is clear that as the batch size becomes smaller, all metrics get better.
For this reason, the best model is the one that is trained using a batch size equal to 32 (16+16), reaching 0.997 of ROC AUC and 0.983 of accuracy.

6.1.4 Impact of the number of the templates

In this section, there is an investigation about how the number of selected templates influences our classification.
Remember that templates are the 1280 numbers extracted by each image belonging to a sub-set of the target dataset. These numbers are the features associated to pictures with people, later compared to features extracted from a new image through the Eucliden distance, in the testing part of *matching*.
Features of images containing individuals should be very similar to templates, in terms of distance, and therefore the related score should be low. On the contrary, features extracted by pictures with other objects should be far from them, producing an high score.
Using a smaller number of templates, means comparing the extracted characteristics with fewer baseline person features.
Perea and Patel in the main article [28] state that reducing templates has no effect on the quality of the classification, if the algorithm is able to extract right features of the class under consideration.
It's time to verify the statement through table 6.4, where all metrics from dataset *BN1* are shown. Values are referred to epoch 400, after a training with λ= 10, batch size= 32, number of target elements = 6000, number of trainable layers= 40.

Performances remain totally unaffected by reducing the number of templates, even in the extreme situation where only one of them is available.
It follows that our Deep One-class Classification for people recognition succeeds in efficiently isolating the target class, extracting representative features of a

N. of templates	AUC	F1 score	Accuracy
40	0.997	0.982	0.983
30	0.997	0.983	0.983
20	0.997	0.983	0.983
10	0.9975	0.983	0.983
5	0.9975	0.983	0.983
1	0.9974	0.982	0.983

TABLE 6.4: Performance metrics of *BN1* by changing the number of templates

person.
Another proof is the visualization of features provided by the dimensionality reduction technique *t-SNE*.

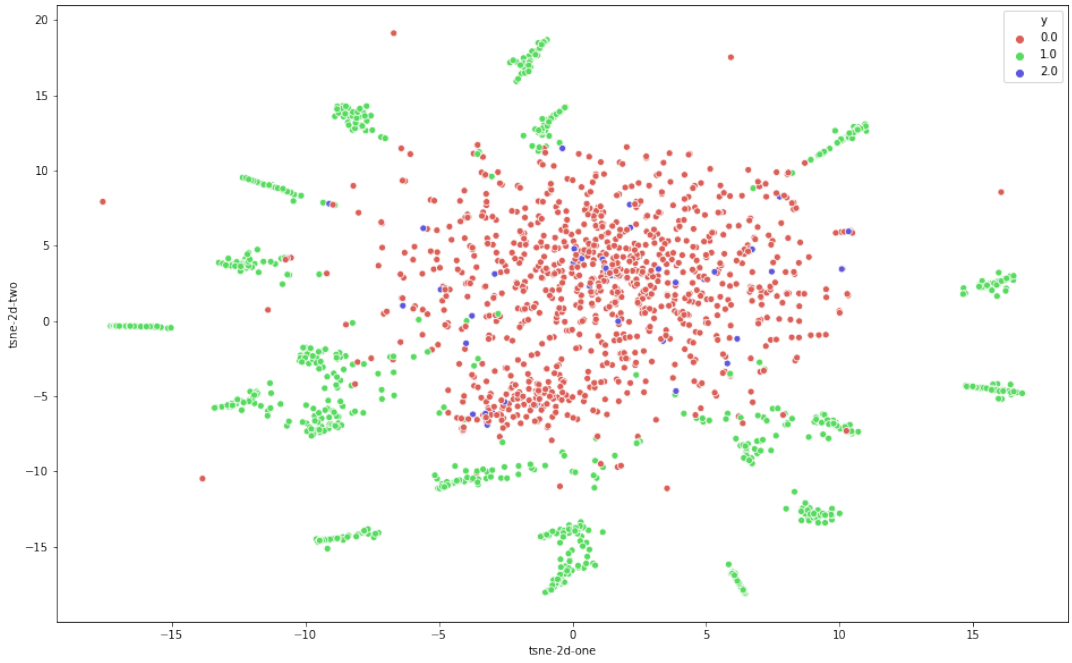


FIGURE 6.5: t-SNE visualization of extracted features and 40 templates

In figures 6.5, 6.6, 6.7, the three cases with 40, 10 and 5 templates available during testing are shown. Inside them, there are:

- *red points* with labels 0, the features associated to images containg people;
- *green points* labeled with 1, the features extracted from pictures with no people;
- *blue points* with a fake label 2, the templates from which the classification score is generated, through the computation of the Euclidean distance among them and features.

In all scenarios, the templates are always in the middle of the target region (red), even when their number is decreased. That's why the reduction has no

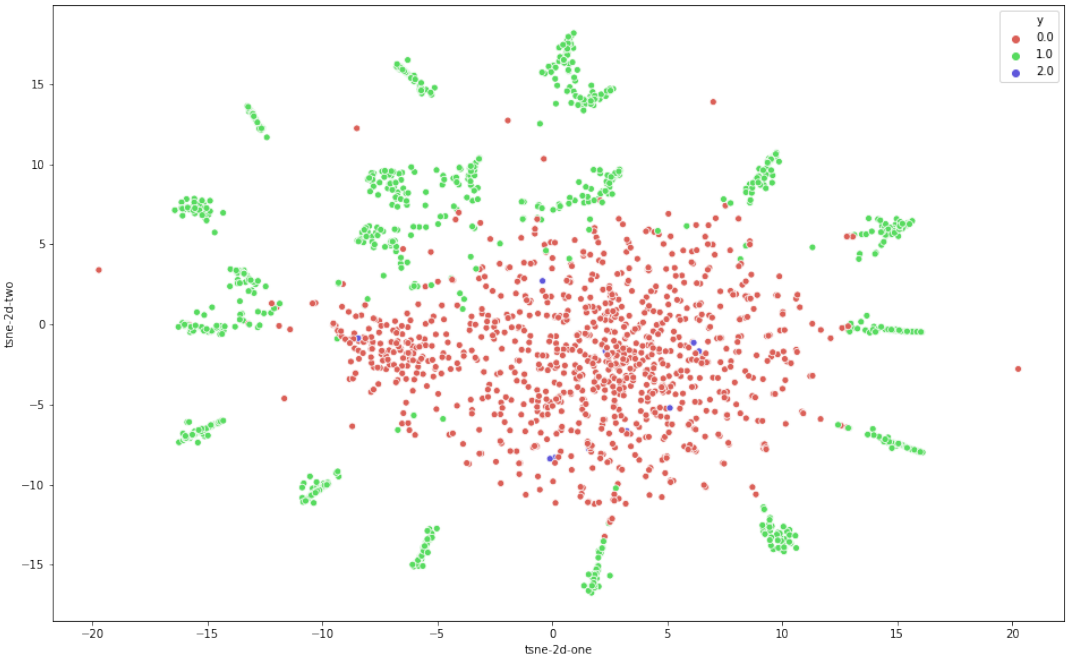


FIGURE 6.6: t-SNE visualization of extracted features and 10 templates

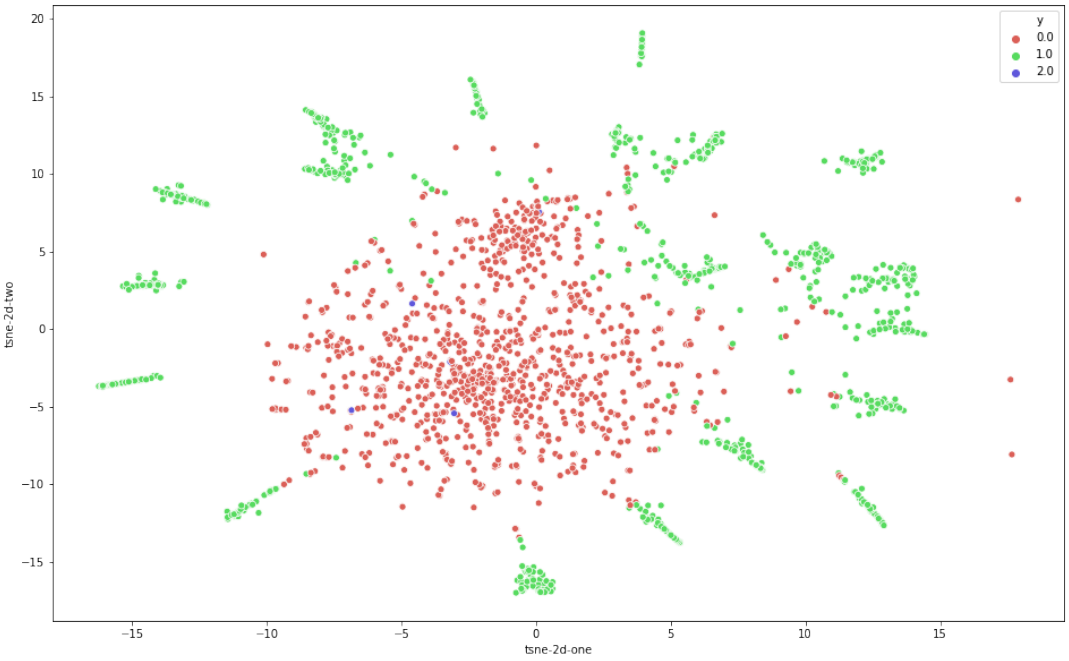


FIGURE 6.7: t-SNE visualization of extracted features and 5 templates

influence on correct classification.
Also, notice that person features are compactly placed together in the red cloud, while features of objects from alien classes are organized into smaller green different regions far from people: our algorithm has reached the capability to produce features with both compactness and descriptiveness properties.

6.1.5 Impact of the number of target examples

In this section, the Deep One-class Classification is evaluated in contrast to binary classification, when the number of people samples available during the training is reduced.
We are aware of the fact that, using the early 6000 elements of the target dataset in the training phase, performances reached by two methods referred to test dataset *BN1* are about the same, like shown in table 6.5.

Method	AUC	Precision	Recall	F1 score	Accuracy
DOC	0.997	0.989	0.976	0.982	0.983
Binary	0.997	0.974	0.986	0.98	0.98

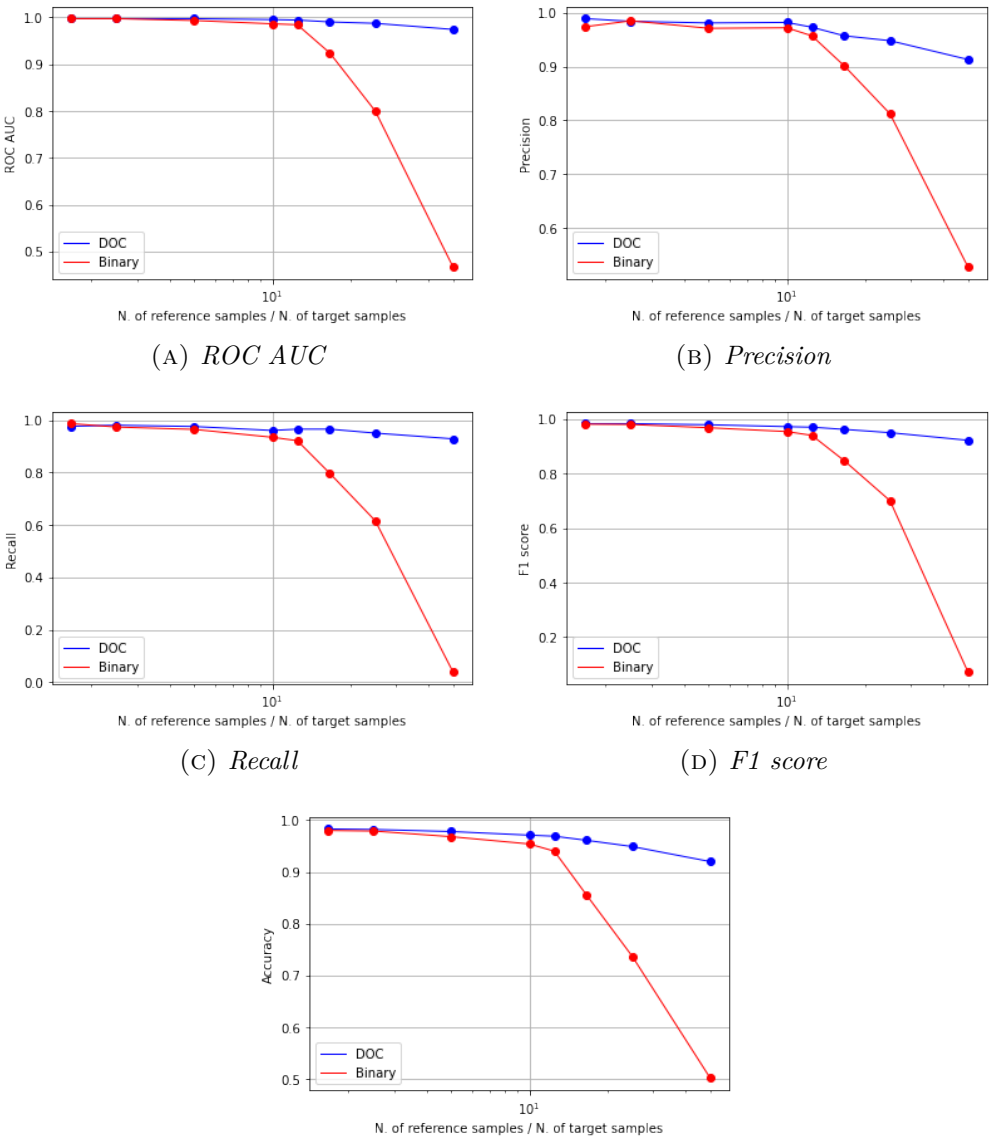
TABLE 6.5: Performance metrics of *BN1* in DOC and in binary classification

Why to prefer DOC to binary strategy if the latter is simpler (no custom compactness and descriptiveness losses, no particular training organization with sub-batches from reference and target elements) and the two solutions produce very similar results? This investigation is carried out precisely to find reasons for choosing our DOC solution.

Results displayed in table 6.5 are absolutely understandble because the dataset employed for training is balanced, since composed of 6000 person images and 10000 outlier images.
The binary classification algorithm is trained in discriminating two classes: *person class*, containing the same examples of the target dataset, and *alien class*, which is filled with images from all categories of the reference dataset.
From its point of view, there are 1.67 alien objects for each target instance in the dataset, so the algorithm has the possibility to fully understand how distinguish what is representing people and what is not representing people, through a wide well-balanced amount of examples.
It is therefore normal that the binary classifier has the same performance results compared to our Deep One-class classifier.

What if we increase the level of imbalance of the training dataset, as suggested in *"One-Class versus Binary Classification: Which and When?"* [47]?
In the article, performances of the two classifiers (OCC and binary) are monitored when the size of the outlier class decreases.
In our application, we cannot modify the number of elements of the negative class because it depends on the external dataset, that should be left unchanged also in a DOC problem with another target class.
We decide therefore to gradually reduce the number of target images of people available during the training.

In the following plots, key metrics *ROC AUC*, *precision*, *recall*, *F1 score*, *accuracy* are shown varying the number of samples in the target dataset, aimed at comparing the two algorithms of Deep One-class Classification and binary classification.
Performances of both models are evaluated on three available dataset *BN1* (fig. 6.8), *BN2* (fig. 6.9) and *IR* (fig. 6.10), reporting values after a training of 400 epochs.



(A) *ROC AUC*

(B) *Precision*

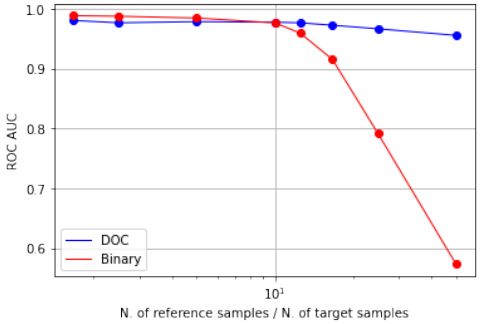
(C) *Recall*

(D) *F1 score*

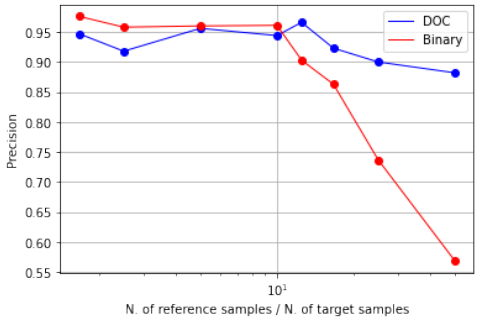
(E) *Accuracy*

FIGURE 6.8: Metrics of testing dataset *BN1* in DOC and binary classification by changing the number of target samples

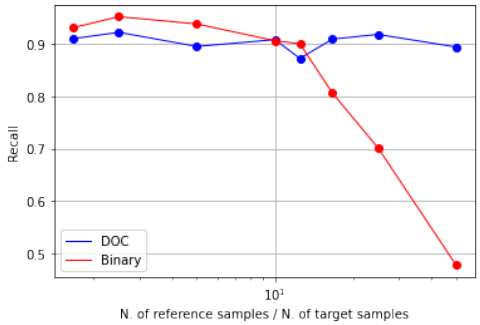
The logarithmic *x-axis* of all plots represents the *ratio* between the number of elements in the reference dataset, fixed to 10000, and the number of examples in the target dataset, that is changing. This choice is taken to properly show the downward trend of performances when decreasing the number of target images.



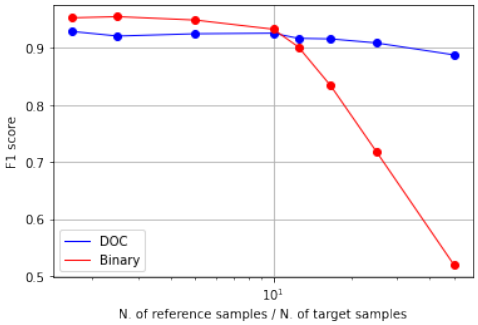
(A) *ROC AUC*



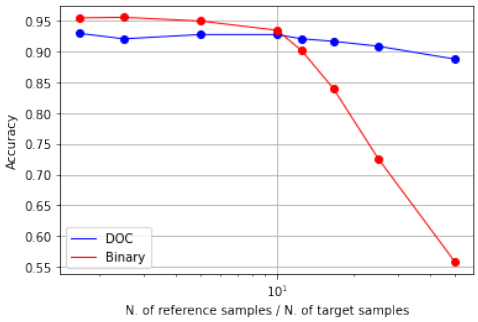
(B) *Precision*



(C) *Recall*



(D) *F1 score*



(E) *Accuracy*

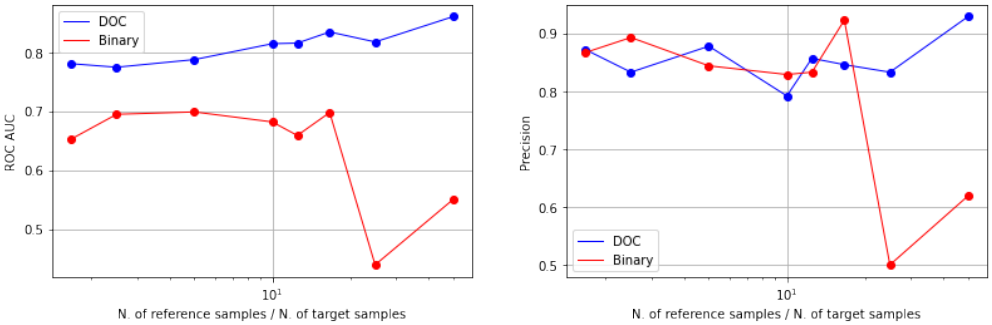
FIGURE 6.9: Metrics of testing dataset *BN2* in DOC and binary classification by changing the number of target samples

In table 6.6, the number of target elements and the corresponding ratio are clarified.

N. of target elements	6000	4000	2000	1000	800	600	400
Ratio ref/targ	1.67	2.5	5	10	12.5	16.67	25

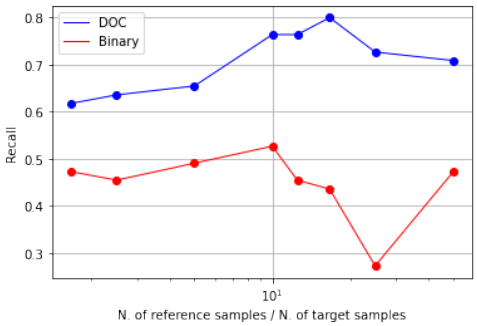
TABLE 6.6: Number of target elements and the corresponding ratio

Remember, also, that in the binary trainings there are the *class person* with the same images of the target dataset and the *alien class* with images from all categories of the reference dataset, and that the input batch size is 32.
The DOC trainings, instead, are characterized by a value of λ consistent with

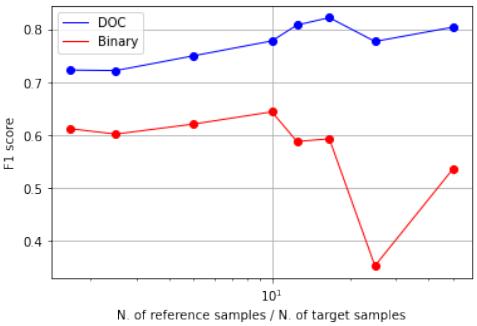


(A) *ROC AUC*

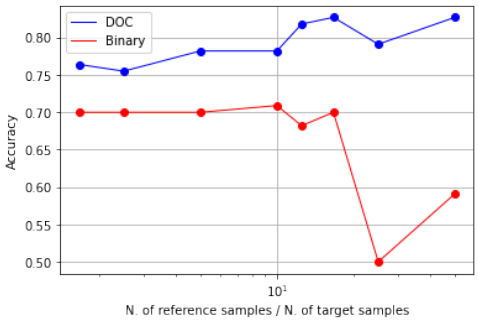
(B) *Precision*



(C) *Recall*



(D) *F1 score*



(E) *Accuracy*

FIGURE 6.10: Metrics of testing dataset *IR* in DOC and binary classification by changing the number of target samples

orders of magnitude of losses, equal to 10, a number of trainable layers equal to 40 and an input batch size of 32 (16 target images + 16 reference images).

All pictures display two lines: the *blue curve* refers to Deep One-class Classification, while the *red curve* relates to binary classification.
Blue and red curves from testing datasets *BN1* and *BN2* are very similar, presenting on average a downward trend, since they are composed of the same type of images used in the training, from *Open Images Dataset V4* and *ILSVRC 2012 dataset*.
Behaviour of metrics coming from testing dataset *IR*, instead, is slightly different: blue curves, in particular, don't decrease when ratio becomes bigger.
This is because InfraRed images are not strictly related to grayscale images

included in the training set, so decreasing people instances inside it reduces the overfitting of the algorithm. Fewer images allow to generalize the concept of person class, that is better adapted to InfraRed images.

Anyway, for all datasets, metrics of the binary classifier (red) degrade, when reducing the number of person images in the training set from 800 on: the binary algorithm is no more able to distinguish people from outliers in the presence of an overabundance of data of the other classes.
Conversely, the Deep One-class classifier still manages to discriminate person instances compared to anything else, by isolating right distinctive features using the previously defined compactness and descriptiveness losses. Performances remain stable with an almost flat curve, even when that number is reduced by 30 times to 200 person images.
The strength of our method is, therefore, the ability to correctly work when the level of imbalance of the training classes increases, with a few hundred of target samples available. Our method is successful also using testing images not directly related to the ones in the training.
In this situations, every binary classifier falls, becoming merely random.

Finally, ROC curves of DOC and binary classification models referred to *BN1* are proposed in figure 6.11, with a different number of person samples in the training.
At the beginning, both ROC curves lie very close to the top left corner of the plot. From the third case on (2000 samples) binary red curve moves away from blue one, confirming to have worse performances.

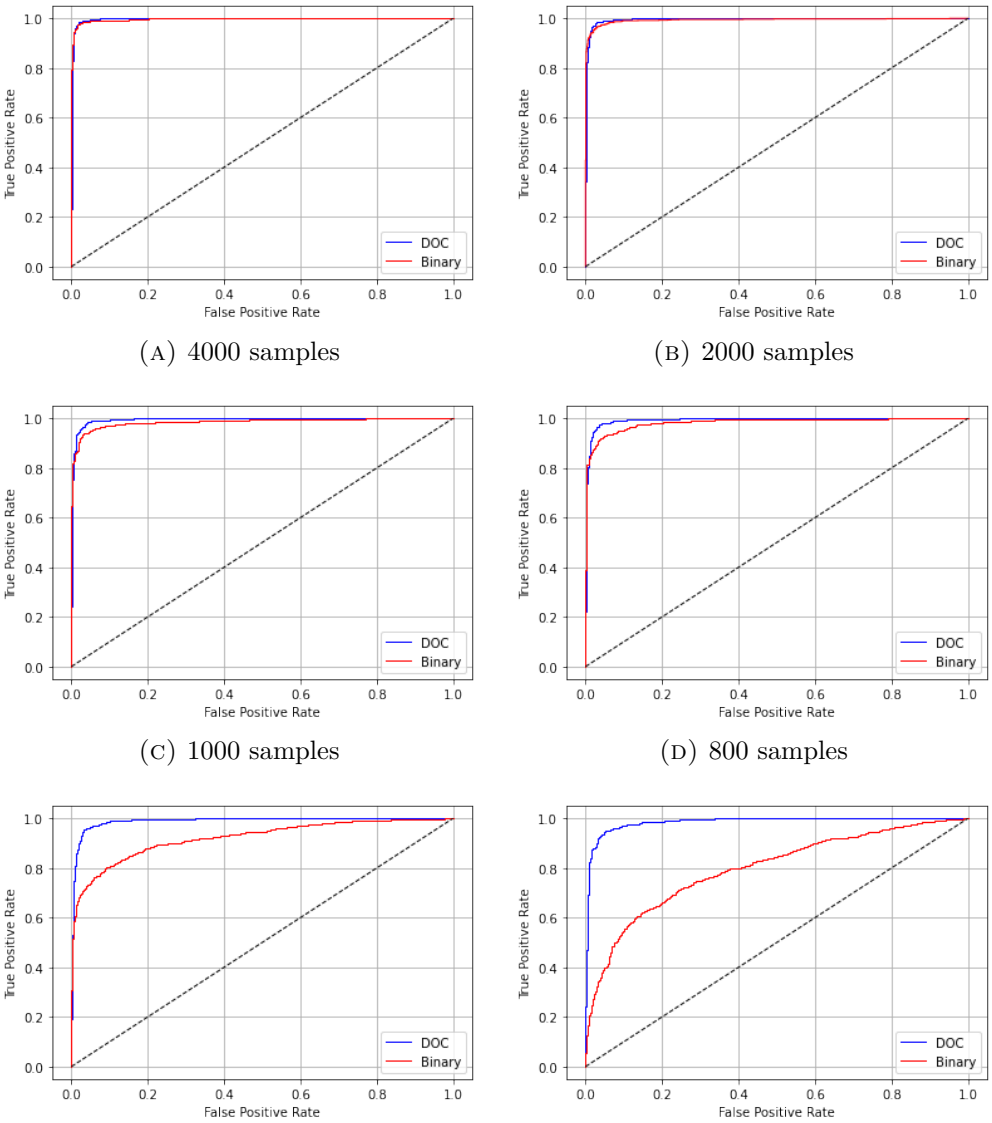
6.2 Conclusions

We have realized, through the method called *Deep One-class Classification*, an effective Deep Learning algorithm for One-Class Classification problems, able to recognize people instances in pictures.
Despite the chosen target class is complicated to model and to be distinguished from anything else, results are promising, showing high values of ROC AUC, F1 score and accuracy.
Our DOC is a very competitive and powerful solution compared to binary algorithms because it successfully works even when unbalanced training dataset are available.

6.2.1 Future work

Suggested next steps to be taken are:

- using another size of images, different from (224,224), to make details of people more visible. Using this size makes the process light and faster, but reduces also the image resolution;



(A) 4000 samples

(B) 2000 samples

(C) 1000 samples

(D) 800 samples

(E) 600 samples

(F) 400 samples

FIGURE 6.11: ROC curves of DOC and binary classification models by changing the number of target samples

- minimizing variance of features, extracting values from another layer. In the discussion we considered the output of the last convolutional block, but it is interesting to see how performances vary using different features;
- employing another reference dataset. We used a subset of ILSVRC 2012, but different external datasets can produce maybe better results;
- modifying the type of network, employing an architecture different from MobileNetV2;
- trying a further transfer learning with the existing InfraRed dataset to tune in a better way parameters;
- acquiring a wider InfraRed dataset to perform the entire training on it.

Acknowledgements

Ringrazio il mio relatore Marcello Chiaberge.
Lei è stato il primo professore con cui ho avuto a che fare approcciandomi al nuovo cammino della magistrale.
Averla come mio relatore è come un chiudersi circolare di questo percorso.
Grazie soprattutto per la disponibilità che mi ha riservato sin da subito e inoltre perché mi ha dato la possibilità, grazie a questo progetto, di esplorare il mondo così intrigante del Machine Learning.

Ringrazio i miei tutor Vittorio, Francesco e Anna.
Il vostro supporto è stato determinante nello sviluppo di questa tesi.
Nonostante la distanza causata dalla pandemia, ho sempre potuto contare sulle vostre delucidazioni e suoi vostri ottimi suggerimenti.
Grazie anche ad Angelo, i tuoi consigli si sono rivelati sempre preziosi.

Grazie mamma e papà.
Siete stati la mia fiamma, il motore del perseguimento dei miei successi universitari. Questa tesi è vostra.
Papà, grazie perché mi hai insegnato cosa significa il duro lavoro e l'ambizione.
Mamma, grazie perché mi hai insegnato che cos'è la determinazione e, soprattutto, l'amore che la vita ti può donare.

Ringrazio Silvia ed Elena.
Grazie Silvia, perché mi hai insegnato, oltre che a cucinare senza olio, cos'è la vera forza di volontà che ti fa scalare perfino le montagne.
Grazie Elena, perché con la tua briosità mi hai sempre spinto a vedere la luce accesa, anche nelle notti più buie.

Ringrazio Michelone.
Tu sognatore, io con i piedi fortemente ancorati a terra. Mi hai insegnato la leggerezza, che non è superficialità ma affrontare la vita con il giusto spirito.
Dal primo esame di Analisi 1 ci sei sempre stato e sono sicura che continueremo ad affrontare giorno per giorno, insieme e con entusiasmo, il futuro.

Ringrazio Chiara.
Mi hai dimostrato che si può voler bene a una persona esterna al pari di un famigliare. Ti ringrazio perché mi conosci per davvero.

Ringrazio tutti i miei familiari per l'appoggio regalatomi in questi anni.
Ci tengo in particolare a ringraziare la nonna, presenza imprescindibile, che non ha mai mancato una "fkezz" a tutti i miei rientri e alle mie partenze, e zio

Franco: finalmente sono una vera 'ingegnera', come dici tu!

Un grazie finale a tutti gli amici che mi sono stati vicini in questi anni: a quelli di Torino con cui ho passato la quotidianità, dalle lezioni alle serate alticce, e a quelli di Barletta, gli amici di sempre, con i quali non ho condiviso i giorni all'università ma le estati e i periodi di festa.

Questo momento segna la fine del mio percorso di magistrale al Politecnico di Torino e sarà sicuramente seguito da un'ondata di cambiamenti importanti.
Il cambiamento è parte della nostra vita e io voglio essere pronta ad abbracciarlo con il sorriso.

Bibliography

[1] Aurlien Géron. *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. 1st. O'Reilly Media, Inc., 2017.

[2] Tom M. Mitchell. *Machine Learning*. New York: McGraw-Hill, 1997.

[3] Favio Vázquez. *A "weird" Introduction to Deep Learning*. URL: https://www.bbvadata.com/a-weird-introduction-to-deep-learning/.

[4] Mohammed Terry-Jack. *Deep Learning: Background Research*. URL: https://mc.ai/deep-learning-background-research/.

[5] Akshay L Chandra. *McCulloch-Pitts Neuron — Mankind's First Mathematical Model Of A Biological Neuron*. URL: https://towardsdatascience.com/mcculloch-pitts-model-5fdf65ac5dd1?gi=814983a68f7d.

[6] [ ] Deep AI. *Perceptron*. URL: https://deepai.org/machine-learning-glossary-and-terms/perceptron.

[7] Michael A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2018. URL: http://neuralnetworksanddeeplearning.com/.

[8] Angelo Tartaglia. *Machine Learning Algorithms for Service Robotics Applications in Precision Agriculture*. Politecnico di Torino, Master Thesis, 2018.

[9] Sagar Sharma. *Activation Functions in Neural Networks*. URL: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6.

[10] Sayali Sonawane. *Activation Function's in Deep Neural Networks*. URL: https://mc.ai/activation-functions-in-deep-neural-networks/.

[11] Saugat Bhattarai. *An Introduction to Gradient Descent*. URL: https://mc.ai/an-introduction-to-gradient-descent-2/.

[12] Andrew Ng. *Deep Learning Specialization - Coursera - offered by deeplearning.ai*. URL: https://www.coursera.org/specializations/deep-learning.

[13] Matthew Stewart. *Simple Introduction to Convolutional Neural Networks*. URL: https://towardsdatascience.com/simple-introduction-to-convolutional-neural-networks-cdf8d3077bac.

[14] datahacker.rs. *Convolutional operation - CNN Convolution On RGB Images - CNN Padding*. URL: http://datahacker.rs/.

[15] aldro61. *Type of learning problems*. URL: https://aldro61.github.io/microbiome-summer-school-2017/sections/basics/.

[16] natasa. *What can we actually do with Deep Learning in Image Processing*. URL: https://www.smartimagingblog.com/category/image-processing-tools/deep_learning/practice/.

[17] Bernhard Schölkopf et al. "Support Vector Method for Novelty Detection". In: (2000).

[18] Drew Wilimitis. *The Kernel Trick in Support Vector Classification*. URL: https://towardsdatascience.com/the-kernel-trick-c98cdbcaeb3f.

[19] D. Tax and R. P. Duin. "Support Vector Data Description". In: *Machine Learning* 54 (2004), pp. 45–66.

[20] Abdollah Dehzangi Akram Vasighizaker Alok Sharma. "A novel one-class classification approach to accurately predict disease-gene association in acute myeloid leukemia cancer". In: (2019).

[21] Michael Kemmler, Erik Rodner, and Joachim Denzler. "One-Class Classification with Gaussian Processes". In: *Computer Vision – ACCV 2010*. Ed. by Ron Kimmel, Reinhard Klette, and Akihiro Sugimoto. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 489–500.

[22] Gert R. G. Lanckriet, Laurent El Ghaoui, and Michael I. Jordan. "Robust novelty detection with single-class MPM". In: *Proceedings of the 15th International Conference on Neural Information Processing Systems* (2002), pp. 929–936.

[23] Lukas Ruff et al. "Deep One-Class Classification". In: ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. Stockholmsmässan, Stockholm Sweden: PMLR, 2018, pp. 4393–4402. URL: http://proceedings.mlr.press/v80/ruff18a.html.

[24] Poojan Oza and Vishal M. Patel. "One-Class Convolutional Neural Network". In: *IEEE Signal Processing Letters* 26 (2019), 277–281. ISSN: 1558-2361. URL: http://dx.doi.org/10.1109/LSP.2018.2889273.

[25] W. Lawson, E. Bekele, and K. Sullivan. "Finding Anomalies with Generative Adversarial Networks for a Patrolbot". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. 2017, pp. 484–485.

[26] Mahdyar Ravanbakhsh et al. "Abnormal Event Detection in Videos using Generative Adversarial Nets". In: *CoRR* abs/1708.09644 (2017).

[27] Mohammad Sabokrou et al. "Fully Convolutional Neural Network for Fast Anomaly Detection in Crowded Scenes". In: *CoRR* abs/1609.00866 (2016).

[28] P. Perera and V. M. Patel. "Learning Deep Features for One-Class Classification". In: *IEEE Transactions on Image Processing* 28.11 (2019), pp. 5450–5463.

[29] Jason Brownlee. *4 Types of Classification Tasks in Machine Learning*. URL: https://machinelearningmastery.com/types-of-classification-in-machine-learning/.

[30] Google. *Colaboratory - Frequently Asked Questions*. URL: https://research.google.com/colaboratory/faq.html.

[31] *jupyter*. URL: https://jupyter.org/.

[32] *Overview of Open Images V4*. URL: https://storage.googleapis.com/openimages/web/factsfigures_v4.html.

[33] Vittorio Mazzia and Angelo Tartaglia. *∼ OIDv4 ToolKit ∼*. URL: https://github.com/EscVM/OIDv4_ToolKit.

[34] Li Fei-Fei et al. *IMAGENET*. URL: http://www.image-net.org/.

[35] Matthijs Hollemans. *MobileNet version 2*. URL: https://machinethink.net/blog/mobilenet-v2/.

[36] Can Pu and Hanz Cuevas Velasquez. *CVonline: Image Databases*. URL: http://homepages.inf.ed.ac.uk/rbf/CVonline/Imagedbase.htm.

[37] S. Kalkan M. Yaman. "An Iterative Adaptive Multi-modal Stereo-vision Method using Mutual Information". In: *Journal of Visual Communication and Image Representation* 26(0):115-131 (2015).

[38] S. Kalkan M. Yaman. "Multimodal Stereo Vision Using Mutual Information with Adaptive Windowing". In: *13th IAPR Conference on Machine Vision and Applications* (2013).

[39] M. Zukal et al. "Interest Points as a Focus Measure in Multi-Spectral Imaging". In: *Radioengineering* (2013), vol. 22, no. 1, pp. 68–81.

[40] Ellmauthaler A., Pagliari C.L., and da Silva E.A.B. et al. "A visible-light and infrared video database for performance evaluation of video/image fusion methods". In: (2019), Multidim Syst Sign Process 30, 119–143.

[41] A. Berg, J. Ahlberg, and M. Felsberg. "A Thermal Object Tracking Benchmark". In: *Advanced Video and Signal Based Surveillance (AVSS), 2015 12th IEEE International Conference on*. 2015.

[42] Simon Lynen and Jan Portmann. *Thermal Infrared Dataset*. URL: https://projects.asl.ethz.ch/datasets/doku.php?id=ir:iricra2014.

[43] J. Davis and M. Keck. "A two-stage approach to person detection in thermal imagery". In: *IEEE OTCBVS WS Series Bench, Workshop on Applications of Computer Vision*. 2005, pp. 4510–4520.

[44] Roland Miezianko. "Terravic Research Infrared Database". In: *IEEE OTCBVS WS Series Bench*.

[45] Jeremy Karnowski. *DETECTION ERROR TRADEOFF (DET) CURVES*. URL: https://jeremykarnowski.wordpress.com/2015/08/07/detection-error-tradeoff-det-curves/.

[46] Luuk Derksen. *Visualising high-dimensional datasets using PCA and t-SNE in Python*. URL: https://towardsdatascience.com/visualising-high-dimensional-datasets-using-pca-and-t-sne-in-python-8ef87e7915b.

[47] C. Bellinger, S. Sharma, and N. Japkowicz. "One-Class versus Binary Classification: Which and When?" In: *2012 11th International Conference on Machine Learning and Applications*. Vol. 2. 2012, pp. 102–106.

[48] Francesco Salvetti. *Image processing algorithms for synthetic image resolution improvement*. Politecnico di Torino, Master Thesis, 2019.

[49] Martin Heller. *Il machine learning, spiegato bene: cos'è, come funziona e quali strumenti si usano*. URL: https://www.cwi.it/tecnologie-emergenti/intelligenza-artificiale/machine-learning-124626.

[50] J. Emile H. Mayo H. Punchihewa and J. Morrison. *History of Machine Learning*. URL: https://www.doc.ic.ac.uk/~jce317/history-machine-learning.html.

[51] *A history of machine learning*. URL: https://cloud.withgoogle.com/build/data-analytics/explore-history-machine-learning.

[52] Dave Myszewski Caroline Clabaugh and Jimmy Pang. *Neural Networks*. URL: https://cs.stanford.edu/people/eroberts/courses/soco/projects/neural-networks/.

[53] Dave Beeman. *Some specific models of artificial neural nets*. URL: http://ecee.colorado.edu/~ecen4831/lectures/NNet2.html.

[54] *History of the Perceptron*. URL: https://web.csulb.edu/~cwallis/artificialn/History.htm.

[55] Adeel Ahmad. *An overview of activation functions used in neural networks*. URL: https://adl1995.github.io/an-overview-of-activation-functions-used-in-neural-networks.html.

[56] Ravindra Parmar. *Common Loss functions in machine learning*. URL: https://towardsdatascience.com/common-loss-functions-in-machine-learning-46af0ffc4d23.

[57] Raúl Gómez blog. *Understanding Categorical Cross-Entropy Loss, Binary Cross-Entropy Loss, Softmax Loss, Logistic Loss, Focal Loss and all those confusing names*. URL: https://gombru.github.io/2018/05/23/cross_entropy_loss/.

[58] Sebastian Ruder. *An overview of gradient descent optimization algorithms*. URL: https://ruder.io/optimizing-gradient-descent/.

[59] aldro61. *Image Classification vs. Object Detection vs. Image Segmentation*. URL: https://medium.com/analytics-vidhya/image-classification-vs-object-detection-vs-image-segmentation-f36db85fe81.

[60] Roemer Vlasveld. *Introduction to One-class Support Vector Machines*. URL: http://rvlasveld.github.io/blog/2013/07/12/introduction-to-one-class-support-vector-machines/.

[61] Shehroz Khan and Michael Madden. "A Survey of Recent Trends in One Class Classification". In: 2009, pp. 188–197.

[62] Rahul_Roy and Akanksha_Rai. *Best Python libraries for Machine Learning*. URL: https://www.geeksforgeeks.org/best-python-libraries-for-machine-learning/.

[63] TensorFlow. *imagenet2012*. URL: https://www.tensorflow.org/datasets/catalog/imagenet2012.

[64] M. Farrajota. *ILSVRC2012 - Imagenet Large Scale Visual Recognition Challenge 2012*. URL: https://dbcollection.readthedocs.io/en/latest/datasets/imagenet.html.

[65] M. Sandler et al. "MobileNetV2: Inverted Residuals and Linear Bottlenecks". In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2018, pp. 4510–4520.

[66] Smita Sinha. *Why Google's MobileNetV2 Is A Revolutionary Next Gen On-Device Computer Vision Network*. URL: https://analyticsindiamag.com/why-googles-mobilenetv2-is-a-revolutionary-next-gen-on-device-computer-vision-network/.

[67] Sik-Ho Tsang. *Review: MobileNetV2 — Light Weight Model (Image Classification)*. URL: https://towardsdatascience.com/review-mobilenetv2-light-weight-model-image-classification-8febb490e61c.

[68] Keras. *MobileNet and MobileNetV2*. URL: https://keras.io/api/applications/mobilenet/#mobilenetv2-function.

[69] Keras. *Transfer learning & fine-tuning*. URL: https://keras.io/guides/transfer_learning/.