UCLouvain

École polytechnique de Louvain

A comparative analysis of deep Re-Identification models for matching pairs of identities

Author: **Nathan Bastien**
Supervisor: **Christophe De Vleeschouwer**
Readers: **Benoît Macq, Gabriel Van Zandycke, Ramin Sadre, Vladimir Sommers**
Academic year 2019–2020
Master [120] in Mathematical Engineering

Abstract

This thesis offers a comparative analysis of deep learning models for Pair Re - Identification. Pair Re-Identification is the task of answering the question *Is this pair of images, each containing a person, from the same person identity ?* This problem differs from usual problems in person Re-Identification where one deals with *finding, in a large gallery of images, the n closest images to a query image.* Models for Re-Identification (Re-ID) handle their task by mapping images into a feature space where each image is represented by an embedding. The distances between the embeddings can then be used as a measure of similarity for Pair Re-ID or to sort the gallery for the usual problem.

The deep learning models implemented for the purposes of this thesis are Convolutional Neural Networks. They draw from existing state of the art models for Re-ID which differ mainly in the supervision with which they are trained. Our models are supervised either by classification or by metric learning or by a combination of both either in succession or simultaneously. Hence the comparative analysis focuses on comparing and contrasting the effects of the supervision on the feature space learned by the models. More precisely on the distribution of distances between embeddings of the same identity and on the distribution of distances between embeddings of different identities. This type of analysis can not be found in the literature.

Acknowledgments

I want to start by thanking my advisor, Christophe De Vleeschouwer, for his continued support during this master's thesis. Our weekly discussions provided tremendous support. I particularly want to thank him for always listening to my interrogations and for answering them with guidance that helped me refocus my thinking many times.

I also want to thank Vladimir Sommers for his advice and for bringing fresh ideas to the table.

Finally I want to thank my family and friends, and in particular Elise, for their love and care during this semester and for listening to my muddled attempts at explanations.

Contents

Introduction

The questions addressed by this thesis fall within the realm of computer vision. Very simply put, computer vision is about using and understanding images with a computer. As one increasingly tends towards automation, that is using machines and computers to solve an always broader area of problems, the challenges of computer vision become increasingly important. Indeed "...apprehension by the senses supplies after all, directly of indirectly, the material of all human knowledge.... It supplies the basis for the whole action of man upon the outer world" (*H. v. Helmholtz*, [1]).

Yet the solutions of some vision problems seem very difficult to describe for computers. Perhaps surprisingly even more so when they are very intuitive. These difficulties originate from that, as stated by *J. Wilding* [2], "What we experience, apparently directly, is actually very different from what is recorded by our sense organs." Moreover "in the case of vision we know very little about how the brain actually processes images" (*K. Dawson Howe*, [3]). This makes it very hard to implement computer programs that mimic our human way of describing the solution to those intuitive vision problems. The *Pair Re-Identification problem* that underpins this thesis is a very good example of this.

1.1 Person Re-Identification

Pair Re-Identification can be formulated as a very simple question. Given a pair of images, each containing a person, the goal is to answer the following question:

Definition (Pair Re-Identification (Re-ID))**.** *Is this pair of images from the same person identity ?*

It seems as a matter of fact a very simple question and is, for us humans, often easy to answer. Indeed when presented with two such images of persons we can more often than not tell if it is the same person in both images. When we recognise different persons, i.e different identities, our decision can most of the time be based on objective, qualifiable, differences such as the colour of the skin, the gender, or the colour of the hair. It can also rarely happen that we are not able to differentiate two different persons. And sometimes, even if we know for sure that it is not the same person in the two images, it is hard to describe how we know it. Our knowledge is intuitive. When we recognise that a pair of images is from the same identity we very often observe the same phenomenon. We intuitively know that we are presented with a single identity but it is very difficult to objectify how we can for sure rule out that it is not someone else.

It was established that the Pair Re-Identification question was, for us humans, mostly easy to answer but that this answer is often intuitive. As mentioned above, this fact makes it hard to write a computer program that is able to do the same thing. The need for such a program can arise in a context of person tracking. The goal of person tracking is to follow the trajectory of one or multiple individuals in a series of images. This

naturally involves differentiating the trajectory of each individual, i.e of each identity. The need to answer the Pair Re-ID question arises when one is not able to follow the trajectory by continuity of movement. This happens very often, because of occlusions from the camera field of view for example.

Even though there are practical applications, we just outlined one, **the Pair Re-ID question is not directly addressed in the literature**, meaning that there is no best approach for writing computer programs that solve it. However it clearly belongs to the broader field of Person Re-Identification (Re-ID). Usually person Re-Identification is concerned with solving the following task:

Definition (Traditional Re-Identification)**.** *Find in a large gallery of images, the n closest images to a query image.*

This problem can arise in a context of public surveillance when the law enforcement spots a criminal at a given place in a city and wants to find other locations this person has visited using the network of security cameras of the city. This task is hard to solve for humans due to the huge quantity of images in the network of security cameras. The appeal for a computer program is thus obvious.

Re-Identification, whether it be Pair Re-ID or Traditional Re-ID, belongs to a broader class of problems called recognition problems. "The term recognition has been used to refer to many different visual capabilities, including identification, categorisation and discrimination. Normally, when we speak of recognising an object we mean that we have successfully categorised as an instance of a particular object class" (*J. Liter et-al*, [4]). Many of the theoretical properties and guiding principles that will be discussed apply for the whole class of recognition problems and will be approached in that context.

State of the art programs that solve recognition problems, including programs for handling Traditional Re-Identification, use deep learning which is a subset of machine learning. Deep learning is the best known solution to intuitive problems such as Pair Re-ID. *I. Goodfellow et-al* [5] defined the solution proposed by deep learning in the following manner:

> This solution is to allow computers to learn from experience and understand the world in terms of a hierarchy of concepts, with each concept defined through its relation to simpler concepts. By gathering knowledge from experience, this approach avoids the need for human operators to formally specify all the knowledge that the computer needs. The hierarchy of concepts enables the computer to learn complicated concepts by building them out of simpler ones. If we draw a graph showing how these concepts are built on top of each other, the graph is deep, with many layers. For this reason, we call this approach deep learning.

Deep learning models for recognition transform images into vector of features called **embeddings** and use those embeddings to solve their assigned task. Features are individual measurable or computable characteristics of objects, persons, or phenomenons.

One can replace the word concept with the word feature in the definition of deep learning by *I. Goodfellow et-al* given above. Deep learning models learn the best feature space for their assigned task. The notion of best feature space is defined by a function called the supervision or loss function.

Deep learning models for Re-Identification differ mainly in their supervision. The different supervisions also result in some changes to the structure of the models. The creation of the best supervision (and of the associated model structure) for Traditional Re-ID is a very active subject of research. There are many recent articles that approach it. However those questions are not approached for Pair Re-ID.

1.2 Research questions

This thesis offers a comparative analysis of deep learning models for Pair Re- Identification, a problem that has not yet been discussed in the literature. The models that have been implemented for the purposes of this analysis draw from existing deep learning models for Traditional Re-ID. Assessing the models for Pair Re-ID enables to discuss and compare the behaviour of the different models on new metrics. It also offers new insights into the effects of the supervision on the feature spaces learned by the deep learning models for Re-Identification.

The questions that are addressed are: How to build the best deep learning model for Pair Re-ID ? Is there a correlation with Traditional Re-ID ? Which behaviours (in terms of characteristics of the learned feature space) of the models are beneficial for those questions ? How are those behaviours related to the supervision ?

1.3 Structure of the thesis

The response to the research questions is organised as follows. The first three chapters of the work offer a strong theoretical background to familiarise the reader with the concepts involved in building a deep learning model for Pair Re-ID. We first introduce machine learning notions and discuss how deep learning has become the preferred approach for solving recognition problems in chapter 2. Then in chapter 3 we look at how and why the data must be organised into a dataset in order for machine learning models to learn from it. Finally we give a detailed overview of the mechanics of deep learning models in chapter 4.

The last three chapter analyse and compare the different models that are implemented to answer the Pair Re-ID question. Chapter 5 gives a complete overview of the models. Chapter 6 then explains the experiments that have been carried out to train the models and analyse their performances. Finally chapter 7 presents the performances of the models, first for Traditional Re-ID then for Pair Re-ID. These performances are analysed through new metrics and examples to outline the behaviours that make a model performant for Pair Re-ID.

Chapter 2

Recognition and Machine Learning

Introduction

Re-Identification (see definition in the introduction), the class of problems of interest in this thesis, belongs to the broader field of recognition. We recall that recognition is used to refer to many visual capabilities including identification, categorisation and discrimination.

The way to solve Re-ID problems is similar in many ways to the way of solving recognition problems in general. This chapter thus focuses on the techniques that helped solve recognition problems with a computer. The preferred method for this task has changed over time and is now to use machine learning. This chapter explains how and why this approach has come to be preferred. It also introduces the reader to the basic notions of machine learning.

2.1 Recognition problems

Traditional algorithms for recognition usually involve matching a known template, or templates, to some extracted part of an unknown image and evaluating the similarity according to a matching criterion. The construction of the template and of the matching criterion range from the relatively simple to the very complicated depending on the algorithm. Some methods, like template matching, involve only a few steps and quite simple mathematical expressions. Others, such as recognition based on the *Scale Invariant Feature Transform* (SIFT), involve extracting complex features through a series of pre-determined steps (gradient analysis, orientation histogram, Hough transform, etc) so that the extracted features possess some desired properties like scale or orientation invariance for example [6]. **Features** are individual measurable or computable characteristics of an object, a person, a phenomenon. The concept of a feature is very important as it is at the core of every method for solving recognition problems.

Even the most sophisticated of those algorithms remained unsatisfactory as they are not very robust to transformations, do not generalise well, and could thus not approach human like performances in identification and in classification [7]. It has been shown by *P. Fischer et-al* [8] that deep learning models (the notion of deep learning is defined in the introduction) outperform SIFT for descriptor extraction and matching by a large margin. Furthermore they are also more robust to all transformations (but blurring). The mechanics of deep learning models are thoroughly described in chapter 4.

2.2 Overview of machine learning

Another approach to solving recognition problems involves using machine learning based models. It is currently the preferred approach for many computer vision problems, including recognition. Indeed, it enables to "tackle tasks that are too difficult to solve with fixed programs written and designed by human beings" (*Goodfellow*, [5]). As was exposed in section 2.1 this description applies to recognition problems.

2.2.1 Models and algorithms

This section starts by defining what a machine learning model is. A machine learning model is a set of mathematical objects (functions, matrices, scalar parameters, etc) that are organised in a given way, i.e **the architecture of the model**, to approximate the process of exactly solving the mathematical representation of a given problem. Every type of model uses a different set of mathematical objects. And every model within a type of models uses a different instance of those objects. For example if some model type uses affine functions $y = ax + b$, the slope a and the intercept b, i.e **the parameters** of the model, will differ between every individual model. The value of the parameters is computed by **learning from the data**. The model learns thanks to one or multiple **algorithms**. An algorithm is a succession of mathematical operations to compute a desired output from a given input, in this case the best value of the parameters based on the data.

2.2.2 Machine learning categories

There are two main families of machine learning models. A model belongs to one or the other family depending on the format of the data it learns from [9]:

- Supervised machine learning: Those models are trained by an algorithm that uses labelled data (training data) to optimise the parameters of a model, i.e to train the model. This model is then used to solve a task on data it has never seen before (testing data). The training takes place beforehand and separately (offline) from the resolution of the problem.
- Unsupervised machine learning: With unsupervised learning models there is no separation between training and testing data and none of the data is labelled. The model learns similarities between the data whilst solving the problem (online) and uses that learned similarity to solve the task it is assigned to.

There are a wide variety of problems that can be solved with machine learning. We define four of them for our purposes:

- Classification: Solving a classification problem means assigning some input data, images for example, to appropriate classes: flowers, cars or persons for example. It is the kind of problem the general public usually associates with machine learning.
- Regression: Regression problems are concerned with predicting a single output from a set of parameters. For example one could want to predict the price of a house from information about its size, location, etc.
- Clustering/Association: Clustering and association regroup problems that consist in finding connections between elements of a dataset and grouping the similar elements together. For example the movie recommendation system on Netflix is based on clustering the users into categories based on the data generated by their behaviour on the platform.
- Metric Learning: It is the task of learning a feature space in which the distances between some input objects have some desired properties. For example one could wish that the distance between images of flowers, in the feature space to which the models maps them, be smaller than the distance between an image of a flower and one of an orange.

Classification, regression and metric learning problems are naturally solved by supervised machine learning. Clustering and association problems are more commonly solved by unsupervised algorithms. However, these are not rigid categories, and one could define many more. Moreover, models usually associated with one category of problems can be used to solve problems commonly considered as belonging to another category.

2.3 Traditional machine learning for recognition

2.3.1 Learning process for recognition

The definition of recognition by *J.C. Liter et-al* given in section 2.1 bears great resemblance with the definition of classification problems of section 2.2. Recognition problems are indeed overwhelmingly solved as classification problems, and thus mainly solved with supervised machine learning models. The models work according to the scheme illustrated in figure 1.

The learning process of models for recognition starts by preprocessing the input pictures/patterns. This means that they modify the image. For example by zooming in on the object of interest, by re-scaling the picture, or by converting it to grayscale. Features are then extracted from those pre-processed images. From there the models learn the best separation between categories of objects in that feature space. The notion of best separation is defined by loss function, also called the supervision of the model. The best separation is the one that minimises that function. Finally they use the optimal separation to classify instances of the objects. The way in which all those steps are

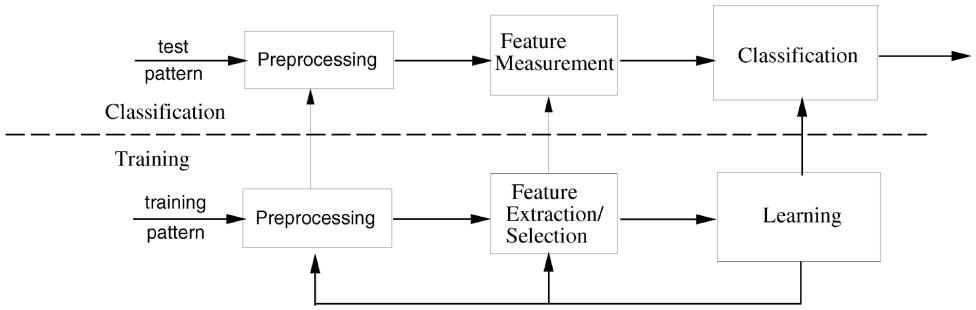


Figure 1: scheme of machine learning models for recognition (*A.K. Jain et-al*, [10])

performed varies from model to model. However, a clear difference can be made between deep learning models and other machine learning models (traditional machine learning) in the way that feature extraction is performed.

2.3.2 Learning complex features with machine learning

Most traditional machine learning models for recognition use hand-crafted or pre-extracted features. For example *Statistical Pattern Recognition* (SPR) for digit recognition uses a few meaningful geometric properties (height, hole area, number of concavities,etc) as features. The *Haar cascade of classifiers* (*Paul Viola et-al*, [11]) uses a combination of many simple features (sums and differences of pixel values in rectangles) to build a robust classifier. The learning part of the algorithm is for those algorithms mainly restricted to the resolution of an optimisation problem to determine the best separation in the extracted feature space between the different classes of the problem.

Linear feature selection However, learning can also incorporate feature selection. This integration was firstly motivated by the fact that "the performance of a classifier depends on the interrelationship between sample sizes, number of features, and classifier complexity" (*A.K. Jain et-al* , [10]). And the dependence is such that the number of training samples is an exponential function of the feature space dimension [12]. This led to the following phenomenon, termed as the *peaking phenomenon* and synthesised by *A.K. Jain* [10]:

> It is well-known that the probability of misclassification of a decision rule does not increase as the number of features increases, as long as the class-conditional densities are completely known (or, equivalently, the number of training samples is arbitrarily large and representative of the underlying densities). However, it has been often observed in practice that the added features may actually degrade the performance of a classifier if the number of training samples that are used to design the classifier is small relative to the

number of features.

The peaking phenomenon led to the use of machine learning to select or extract better features from the pre-extracted features. For example it is performed in the construction of the *Haar classifier* via the incorporation of *AdaBoost* (*Yoav Freund et-al*, [13]).

The most well known and used technique for feature selection is probably the *Principal Component Analysis (PCA)*. It is used in conjunction with many recognition algorithms. Its goal is to select the most expressive features through a linear transformation of the original feature space based on an eigenvalue analysis of the covariance matrix of the training patterns. PCA is an unsupervised learning technique but there are also supervised linear transformation techniques, such as *Fisher's Linear Discriminant Analysis*, that uses the class information of the training patterns to build features that are explicitly trained to separate the classes. The feature construction in all these linear algorithms can be summarised by the following equation:

$$Y = XH \tag{1}$$

where $X \in \mathbb{R}^{nxd}$ is the original pattern matrix, $H \in \mathbb{R}^{dxm}(m < d)$ is the transformation matrix and $Y \in \mathbb{R}^{nxm}$ is the new feature matrix. The construction of the matrix H varies from algorithm to algorithm.

Non-linear features The principal limitation of those methods is that they can only learn linear transformations. If the separations between classes is non linear, it will struggle. This limitation can be partly solved by the introduction of the *kernel trick*, summarised by *A.K. Jain* [10]:

> The basic idea of [the kernel trick] is to first map input data into some new feature space F typically via a nonlinear function ϕ (e.g., polynomial of degree p) and then perform a [linear analysis] in the mapped space. However, the F space often has a very high dimension. To avoid computing the mapping *phi* explicitly, [kernel trick based methods] employ only Mercer kernels which can be decomposed into a dot product:
>
> $$K(x, y) = \phi(x)\dot{\phi}(y) \tag{2}$$
>
> As a result, the kernel space has a well-defined metric. Examples of Mercer kernels include pth-order polynomial $(x - y)^p$ and Gaussian kernel $e^{||x-y||^2}$.

The transformation matrix in the kernel space F, similar to the matrix H of equation 1 is built using the function K of equation 2. The major problem of this approach is **that the choice of the kernel function is arbitrary and the result of an intuitive decision of the machine learning engineer**.

2.4 Deep Learning for recognition

Deep learning, of which the basic principle was described in the introduction, is a subset of machine learning that pushes even further the concept of extracting and selecting the best features introduced by techniques such as PCA. Deep learning models are called neural networks, because their architecture is inspired by the human brain. Neural Networks for visual processing and thus for recognition are called Convolutional Neural Networks (CNNs). The mechanics of Neural Networks are the subject of chapter 4.

Convolutional Neural Networks also function according to the scheme represented on figure 1 page 7. A key difference is that those models learn the best features and the optimal classification together. This principle is called *end to end learning*. Indeed, CNNs do not start to learn from pre-extracted features but from the images/patterns themselves after an eventual preprocessing step. The input features are the values of every pixel in every colour channel. They do not require to choose any arbitrary features or kernel functions beforehand. Figure 2 shows a representation of the learning scheme of CNNs. It can be seen as a CNN version of figure 1.

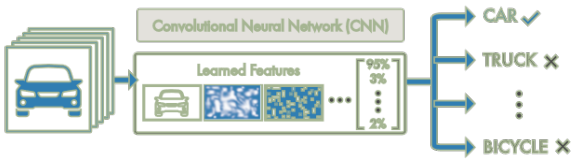


Figure 2: Deep learning scheme for recognition

Convolutional Neural Networks outperform other machine learning models for recognition by a large margin and are thus the overly preferred method for those tasks. The results of the *ImageNet Large Scale Visual Recognition Challenge (ILSVRC)* are a very good indicator of the impact and performance gain that resulted from the use of CNNs and of their improvement over time. This challenge is indeed "a benchmark in object category classification and detection on hundreds of object categories and millions of images" (Olga Russakovsky, [14]). It was held from 2010 to 2017. The evolution of the top-5 classification error from the winning model is displayed in figure 3.

One can see that the introduction of deep learning models drastically improved the performance. In less than five years deep learning models have improved to the point that their performances are comparable with those of humans. Traditional computer vision models, even when enhanced with other machine learning algorithms, could not get close to that range of performances.

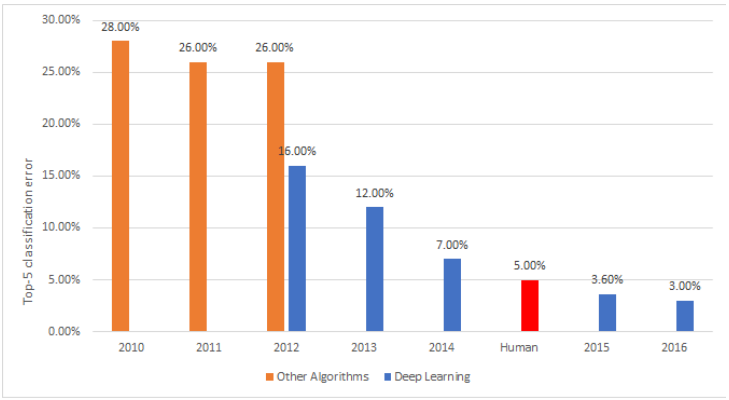


Figure 3: ILSVRC results evolution [14] [15]

Summary

This chapter started by discussing the guiding principles of solving recognition problems, the class of problems Re-Identification belongs to. It then outlined why machine learning became the preferred approach for solving those problems. It defined the general scheme machine learning models use to solve recognition problems, a scheme based on learning the best separation between categories of objects in a feature space. It then explained how the models gradually started to learn this feature space as well as the separation between the categories. Finally it showed how deep learning models have immensely improved performances in the field by learning very complex feature spaces.

Chapter 3

Construction of a dataset from raw data

Introduction

It was stated in chapter 2 that the re-identification problem, defined in the introduction, is a recognition problem. We then explained that the preferred approach for solving recognition problems is to build deep learning models that extract discriminative features from the data and use those features for a given task, classification for example. Those features are extracted from the data. But what does from the data means ? What do we call data in that context ? How can we use it to train a machine learning model ?

These are the, often overlooked, questions this chapter will discuss. We will see that sourcing raw data is far from sufficient for machine learning purposes. The data also needs to be checked and prepared, labelled, and organised in order for machine learning models to learn. This whole process is called the **dataset construction**. It is often an arduous task as the raw data one wishes to use is often not collected with machine learning purposes in mind or not collected by specialists. This results in an enormous amount of work for data scientists. It is estimated that between 70% and 80% of an analysis cycle is devoted to dataset construction (*Encyclopaedia of machine learning and data mining*, [16]).

3.1 Data in the context of Machine Learning

Because every machine learning model originates from the data, this section starts by defining what data is. Data is a very generic term. It is defined by *the Cambridge Advanced Learner's Dictionary* [17] as "information, especially facts or numbers, collected to be examined and considered and used to help decision-making, or information in an electronic form that can be stored and used by a computer."

In the context of machine learning one needs to introduce a preciser definition. The data, as in "the data used to train a machine learning model", is a set of **samples (data points) from an underlying distribution**. This terminology comes from the field of probability and statistics where a distribution is a function that lists all possible values that the data can take. Think for example of the Gaussian distribution that is used in many different domains to represent real valued random variables. Those functions have a mathematical expression such as $h(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}$ for the 1D Gaussian distribution.

In the field of machine learning the distributions do not have known mathematical expressions. We assume that one exists but that we do not and can not known it, hence

the qualifier underlying. The inputs can be pretty much every type of data, pictures for example.

The goal of training a machine learning model on the data points one can source is that the model will "understand" the data, that its parameters will be optimised in regard of the underlying distribution the data comes from. One can then use this understanding to solve the problem he build the model for on other data points from the same distribution.

However, if one just feeds raw data to the model it will not learn anything, or not nearly enough, from the underlying distribution. It may only learn specific characteristics of the data points used for training. This is called overfitting on the training data. In such cases the model will not generalise to data points outside the ones it was trained on and will thus be effectively useless. This explains the motivation and the importance of constructing a dataset. Now that we have defined what data is and in doing so the motivation for the dataset construction we will take a look at the different steps involved in that process.

3.2 Ensuring the quality of the data

It is clear from section 3.1 that data is the basis of every machine learning model. If one wants his model to be qualitative he should make sure that the data he uses is qualitative. The dataset construction starts thus by evaluating the data one has at his disposition. As a result some of the data will be deemed unfit for learning purposes. This section will describe this data evaluation. It is strongly inspired by the entry on *Data preparation* from the *Encyclopaedia of machine learning and data mining* [16].

The evaluation work consists of multiple steps. The thesis will not go into details on the subject because the evaluation of the data for the dataset construction was not part of the work. However, the key steps are mentioned and illustrated below. For a detail overview the reader is referred to the *Encyclopaedia of machine learning and data mining* [16].

The first step is to define what is expected of the dataset. For that we can define quality measures as on the figure 4.

Then one can analyse its data according to those quality measures and update the dataset accordingly. This is done through an iterative process called **data cleansing** which is illustrated on figure 5. The *workflow* consists in eliminating the data non compliant with the quality measures.

After this step one can add some specific data points to the dataset if some key part of the distribution is missing, this is called **data enrichment**. Finally one can choose to modify the representation of the data, for example by resizing all pictures. This last step is called **data transformation**.

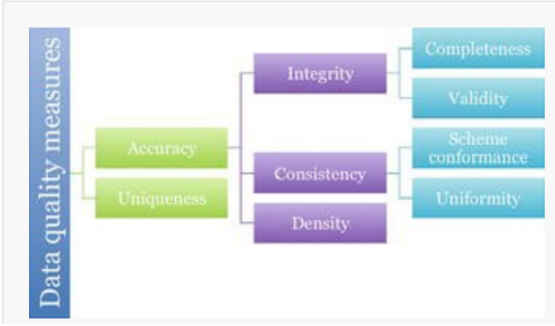


Figure 4: Data quality measures adapted from [18] in [16]

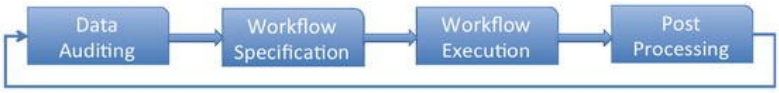


Figure 5: Data cleansing process adapted from [18] in [16]

3.3 Ground truth and labelling the data

This section assumes that the raw data has been evaluated and reduced to useful data points. If one wishes to use this data for unsupervised learning (definition in section 2.2), it may be enough. On the contrary, if the model that will use the data belongs to the family of supervised learning models (again see section 2.2), the dataset construction is far from over. Indeed, models generated through supervised learning **improve by using labels assigned to the data**.

An important part of the dataset construction is to generate those labels. This creation process, whatever the form of the labels, is called the ground truth creation. It is a very important step because the model learns against this ground truth and if it skewed so will the resulting model. Because machine learning applications, and thus the type of ground truth that can be created, are very broad we will restrict the rest of this discussion on ground truth to machine learning applied to computer vision problems.

The objective of any computer vision system is to extract some information from images. The ground truth creation thus mainly consists in assigning some truth value to pixels or groups of pixels that make up the images. Sometimes this seems easy enough. For example, while generating ground truth for a recognition model that needs to distinguish apples from oranges, one simply needs to label training images of fruit as apple or oranges. It is very intuitive and objective, everyone is in agreement and the model should perform as expected in every situation. On the contrary when designing a model that localises the edges on an image it is very difficult. There is no global agreement on what an edge is and the viewpoint of the ground truth creator is thus subjective. This subjectivity will greatly impact the model performances.

For recognition problems, as defined in section 2.1, the ground truth creation consists

in assigning one of the categories represented in the training data to every image. The example of apples and oranges given above is one such problem. When one only possesses images that contain solely instances of the categories to classify it is very easy. On the contrary, if images contain more than the instances, if they are in a (complex) environment, it can become very difficult. The ground truth creator must Indeed, restrict as well as possible the image to the instance. Otherwise, if the environment of the training data images is not diverse enough or if it is radically different from the environment of the testing data images or of the real data, the model will perform badly.

Indeed, it is very possible that the model will have learn to not only recognise the categories but also the environment around the instances if not the environment only. This outlines the importance of restricting the environment as much as possible or to ensure an important diversity of environment. Both can be equally difficult.

Ensuring the diversity of environment requires a huge quantity of data. Sourcing so much data leads to numerous difficulties : Ethical and legal questions about data privacy, important financial cost, uncertainty about the continuity of the source of the data, etc (*Ben Lorica*, [19]).

Restricting the image to the instance itself can be labour costly. The most used method is to isolate the instance in a bounding box that can also contain a small part of the environment. Those boxes are often created by an object detector program (that can be computationally costly) in combination with some hand drawing (that is labour greedy). Figure 6 shows an example of bounding boxes that can be used for recognition tasks.

Figure 6: An example of bounding boxes from the *CAVIAR* dataset [20]

3.4 Model evaluation and organising the data

As mentioned in section 3.1, the aim of training a machine learning model is that the parameters of the model will be optimised in regard of the underlying distribution the training data comes from. In the case of supervised machine learning models one

can, and should, use part of the data he possesses to verify that the model learned the underlying distribution characteristics and not some specificity of the examples it was trained on. If the model learned from the distribution we say that it generalises well.

Training set To ensure that the model generalises one should split its data in three sets. The first one is the training set. It contains the data that will be used to optimise the parameters of the model. Training set data is labelled and, for deep learning purposes at least, should contain as much data as possible. The more training data the better seems actually pretty obvious for any machine learning model because it provides the opportunity to learn from more examples, and thus for more robust learning. However, it has been showed that machine learning techniques other than deep learning do not scale with the amount of data. This is illustrated in figure 7. This fact is one of the reasons behind the recent success of deep learning.

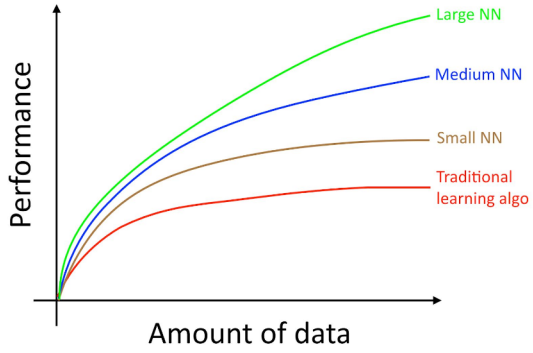


Figure 7: How machine learning models scale [21]

Testing set The testing set contains data points from the same underlying distribution as the training set but that are not use to train the model. It serves as an **unbiased evaluation of the performances of the model**. It shows how well the model generalises. This means that a machine learning engineer should not use the results on the testing set as a basis to improve the model. There was a rule of thumb that the partition of data between training and testing should be 70%-30%. But, with the emergence of *Big Data* the relative size of the testing set has shrunk. It should only be large enough for a fair performance analysis (*Andrew Ng*, [21]).

Validation set The last set is the validation set or development set. It is similar to the testing set in that it contains data points from the same underlying distribution as the training set and is also used to evaluate the performances of the model. But the evaluation on the validation set is used to tune the model for better performances. It is a biased evaluation. The data points in the validation set are often closer to the training

set points that the testing set points are. Indeed, the data is often first separated into training and testing set. Then a fraction of the training set data is reserved to be used for validation.

The model sees this data while training but does not learn from it. The results on this set are **the best indicator of the quality of the training**. It can for example show if a model will generalise well to data points outside the training set. There is no fixed rule for the relative size of the validation set compared to the training set size. The important is that it contains multiple examples from all the training set categories.

Summary

This chapter explained that in order to create a good machine learning model one first needs to construct a good dataset. The raw data needs to be evaluated, labelled and organised to form the dataset. All these steps require human intervention and subjective decisions need to made. Those decisions have an impact on the performances of the machine learning model and thus should not be overlooked. The construction of the model, the design of its training scheme, etc (chapter 4 focuses on those questions for deep learning models) often attract more attention. It is completely understandable as they are mathematically more complex and posses the potential for bigger breakthroughs. Yet the reader is advised to keep those questions in mind as we turn to models related questions in the next two chapters.

Chapter 4

Deep Learning overview

Introduction

This chapter is intended as a practical introduction to the way deep learning models are built to solve recognition problems. Chapter 2 explained why deep learning models are now the preferred method to solve recognition problems including Re-Identification. The goal of this thesis is to offer a comparative analysis of such models for Pair Re-ID (Pair Re-ID is defined in the introduction). It is thus very important to offer the reader an overview of deep learning models, of how they learn and of the associated challenges, in order to appreciate the discussion of the subsequent chapters.

This chapter does not aim to cover the whole discussion around deep learning but to give the necessary knowledge to understand the concepts discussed in the rest of the thesis. For a detailed discussion around deep learning the reader is referred to *Deep learning* by *Ian Goodfellow et-al* [5] from which this chapter draws significantly.

4.1 Neural Networks

4.1.1 Neuron and Network

There are multiple kinds of deep learning models but they are all neural networks. Defining the concepts that lie beneath those two words, neural and network, is key to understand how deep learning models work.

Neural networks are composed of **neuron units**. A neuron unit is simply a function f, called the activation function, which is given a linear combination of values as input. Equation 3 gives a generic expression of the output of a neuron:

$$u = f(w_0 + \sum_{i=1}^{p} w_i x_i) \tag{3}$$

where the p input values are x_i and the weights are w_i. The weight w_0 is associated with a constant input value of 1 and called the **bias** of the unit. The value $z = w_0 + \sum_{i=1}^{p} w_i x_i$ is called the **logit** of the unit. A neuron is said to be active when the value of its logit is positive.

A neural **network** is the composition of neuron units. This network is "represented by a direct acyclic graph describing how the functions are composed together" (*I. Goodfellow et-al*, [5]). The neurons are organised into layers. The first layer represents the inputs of the model, the last one represents the outputs of the model, and all the layers

in between are **hidden layers**. The outputs of the units of an hidden layer are fed as input to the units of the next layer. If all the outputs of the previous layer are fed to all the units of the next one, this next layer is called a **Fully-Connected layer (FC)**. Equation 4 gives the mathematical relationship between a FC layer and the layer preceding it:

$$U_n = f(W_n U_{n-1} + b_n) \tag{4}$$

where W_n contains the weights w_i and b_n the biasses w_0.

A network that only consists of fully connected layers is a **Multi-Layer Perceptron (MLP)**. The number of hidden layers of a deep learning model is called the **depth** of the model. An example of a graph representing an MLP of depth 2 is given in figure 8.

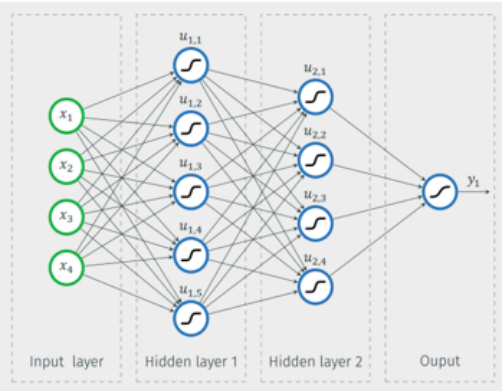


Figure 8: A simple Multi-Layer Perceptron (reproduced from [22] with permission)

Combining neuron units together into a network is equivalent to an organised composition of their functions. The resulting function will be denoted $h_\omega(x)$. It is a very complex function of the inputs x and of the set of weights ω of the network. The idea behind deep learning is to learn an optimal set of weights ω so that $h_\omega(x)$ will perform the task the model was built for as well as possible. How a network learns the best set of weights ω will be explained in section 4.3.

Mathematically the notion of the network performing its task as well as possible is translated by approximating with $h_w(x)$ a "perfect" function, denoted $h^*(x)$, which would solve the mathematical representation of the problem. This function $h^*(x)$ has most of the time no expression. The rest of this section will introduce and discuss two notions that are key to ensure that the model produces a good approximation: the choice of the functions of the units and the organisation of the network, i.e its architecture.

4.1.2 Activation functions

The choice of the activation function for a given neuron differs whether the unit belongs to the output layer or to one of the hidden layers. The functions are often defined for a whole layer. Theoretically every function could be an activation function

and their design is "... an extremely active area of research and does not yet have many definitive guiding theoretical principles" (*Goodfellow et-al*, [5]). However, in practice only a small number of families of functions are used. The by far most common family is the one of rectified linear or ($ReLU$) functions. Equation 5 gives a generic expression of the ReLU functions:

$$f(z) : f_i(z) = \max(0, z_i) + \alpha_i \min(0, z_i) \quad i = 1...m \tag{5}$$

where $z, f(z) \in \mathbb{R}^m$ with m the number of units in an hidden layer. $f(z)$ is thus the output of the whole layer and z is the vector containing all the logits of the layer. In the most common case the value of α_i is 0 for all i. The popularity of the ReLU family comes from the fact that they perform well in almost every case. It has even been shown that "using a rectifying nonlinearity [ReLU function] is the single most important factor in improving the performance of a recognition system" (*Jarrett et al.*, [23]).

The choice of the activation function for the output neurons depends on the problem one wants to solve. It will determine the number of output values and sometimes constraint the values the output can take. For example the output of a model for classification should be a probability distribution over m classes. The output layer should have m units and the output of each neuron of the layer should be a number between 0 and 1, with all the outputs adding up to 1. To ensure this a function called *softmax* is predominantly used. Equation 21 gives the expression of one of the outputs of a softmax layer:

$$u_i = \text{softmax}_i = \frac{\exp z_i}{\sum_{j=1}^{m} \exp z_j} \tag{6}$$

4.1.3 Architecture and depth

The architecture of a network determines how its neuron units are composed together to create the approximation $h_\omega(x)$. It seems thus a key question at first sight. However, in theory a network with a single hidden layer should be sufficient to approximate any function $h^*(x)$. Indeed, **the universal approximation theorem** (Hornik, 1991, [24]) states that

> A feedforward network with a linear output layer and at least one hidden layer ... can approximate any ... function from one finite-dimensional space to another with any desired nonzero amount of error, provided that the network is given enough hidden units. The derivatives of the feedforward network can also approximate the derivatives of the function arbitrarily well.

This means that, whatever the perfect function $h^*(x)$ one is trying to learn, there exists a MLP that can represent it. But it does not mean that the training algorithm will be able to learn this particular MLP. It actually turns out that the architecture has a great importance with regard to how well the training algorithm can learn an approximation $h_\omega(x)$.

Up to the 2000's Multi-layer Perceptrons were predominantly used and the most discussed architecture design question was the depth. "In many circumstances, using deeper models can reduce the number of units required to represent the desired function and can reduce the amount of generalisation error" [5]. From the 2000's onward specific architectures have been created to better fit the different kinds of problems and to overcome difficulties that occurred during training. Section 4.2 describes the architecture of the models used for image processing.

4.2 Convolutional Neural Networks

4.2.1 Definition and inspirations

One of the fields where deep learning first gained momentum was image processing and computer vision, and especially recognition (recognition problems are defined in the introduction). Because Re-Identification is a subset of recognition, models for Re-ID use a same type of architecture as all the models for recognition of images: *Convolutional Neural Networks (CNNs)*.

The architecture of CNNs has two main inspirations. The first one is biological. Indeed, Convolutional Neural Networks are inspired by the primary visual cortex. The primary visual cortex is the first region of the brain to "perform significantly advanced processing of visual input" (*Goodfellow et-al*, [5]). It is organised as a succession of operations on two-dimensional spatial maps. The layers of CNNs are organised as two-dimensional maps of units to mimic that structure.

The second one is mathematical. CNNs are indeed based on an operation called convolution (hence their name) which is a fundamental operation in many signal processing systems, including in image processing. In image processing the input signals of the convolution are often an image channel, that is a 2-dimensional array of pixels values, and a kernel, which is a smaller two-dimensional matrix. The result of the convolution between an image channel $I(i, j)$ and a kernel $K(m, n)$ is given by equation 7:

$$S(i,j) = (I * K)(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n) \tag{7}$$

When an image has multiple channels, such as colour images which typically have three (Red Green and Blue), the result of the convolution between image and kernel is $\sum_{c=1}^{3} K * I_c$. CNNs use the convolution operation to go from one hidden layer to the next at least once. The convolution links layer $n-1$ and layer n replacing the matrix operation of equation 4. The elements of the kernels are the set of weights ω learned by the training procedure of the network.

Convolutional Neural Networks are thus neural networks whose architectures and properties are inspired by the primary visual cortex and where the operation from one layer to the next is one or multiple convolutions. If multiple kernels are applied on one

layer, the next layer will have as many channels as there were kernels. The dimensions of the layers in a CNN are thus triplets (width, height, channel).

As was discussed in section 2.4 the goal of deep learning models for recognition is to learn discriminative features from the images. Those features are computed by going through the CNN. The first CNN that reached high level performance is named *LeNet-5* and was introduced by *LeCun et-al* in 1998 [25]. The process of going through the CNN from the image to the features is called the **forward propagation** and can be represented schematically as in figure 9.

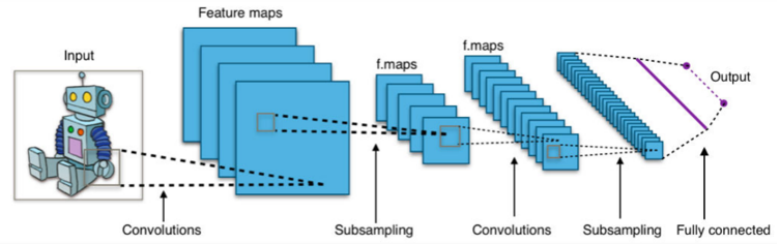


Figure 9: Schema of the forward propagation in a CNN (reproduced from [22] with permission)

4.2.2 Forward propagation

The forward propagation transforms a three channel image into a feature vector by using multiple operations. The remainder of this section describes those operations mathematically.

Convolution A key operation is the convolution. It was described generically in equation 7 but we will now focus on its application in a CNN. The output of a unit at spatial position (i, j) on channel k of convolution layer $U_t \in \mathbb{R}^{W_t \times H_t \times C_t}$ is given by equation 8:

$$u_{ijkt} = f((U_{t-1} * K_{kt})(i,j)) = f(w_{0kt} + \sum_{m=\frac{-M}{2}}^{\frac{M}{2}} \sum_{n=\frac{-N}{2}}^{\frac{N}{2}} \sum_{c=1}^{C_{t-1}} w_{mnkt}\ u_{(i-m)(j-n)c(t-1)}) \quad (8)$$

It is the result at position (i, j) of the convolution of kernel $K_{kt} \in \mathbb{R}^{M \times N}$ with layer $U_{t-1} \in \mathbb{R}^{W_{t-1} \times H_{t-1} \times C_{t-1}}$.

The weights w_{mnkt} are the entries of the kernel matrix K_{kt} and w_{0kt} is the bias of the layer. The kernel K_{kt} is shared between all positions (i, j). Its application will result in one channel of layer U_t. The fact that the kernel is shared between all spatial positions is called parameter sharing. It is a major difference with Multi-layer Perceptrons because rather "than learning a separate set of parameters for every location, we learn only one

set" (*Goodfellow et-al*, [5]). It thus reduces the storage needs for the same number of units.

The mathematical expression of the convolution in a CNN given by equation 8 might seem overwhelming at first and involves a lot of indices. This complexity does not come from the convolution operation, which is a simple matrix operation, but from the structure of the CNN. If we do not consider all structure related indices and parameters, the convolution between one layer and the next is a simple as the schema in figure 10.

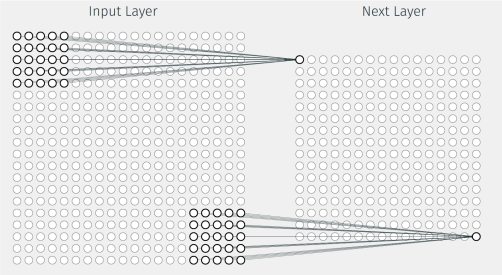


Figure 10: Convolution between succeeding layers (reproduced from [22] with permission)

Padding Questions can arise regarding what happens for the boundary units of layer U_t during convolution. Indeed, for coordinates (i, j) of layer U_t that correspond to boundary coordinates of layer U_{t-1} the convolution operation involves units of layer U_{t-1} that do not exist. The operation that deals with this issue is called *padding*. There are two main padding strategies. The first one is to define imaginary units with value zero outside the boundary of layer U_{t-1}. This strategy is called zero padding or same padding. It results in layers U_t and U_{t-1} having the same width W and height H. The second strategy is simply not to compute the units of layer U_t which need neighbours outside the domain of layer U_{t-1}. This is called valid padding and results in a reduction of the width and height of layer U_t compared with layer U_{t-1}. We have $W_t \neq W_{t-1}$ and $H_t \neq H_{t-1}$.

Sub-sampling The effect of spatial domain reduction is a side effect of the convolution and can be neutralised using zero padding. However, reducing the width and height of the layers as we move forward into a CNN is desired. It is called sub-sampling and often goes along with an increase of the number of channels. There are two operations that are used to reduce the size of the spatial domain from one layer to the next.

The first one is performed on convolution layers and consists in augmenting the **stride** of the layer. By default a convolution layer has a stride of one. This means that the convolution operation will be applied for every possible pair of coordinates on the domain $W \times H$ of layer U_{t-1}. If the stride of the layer is two, the convolution will only be applied for one width and one height coordinate in two. This means that we will have

four times less units in layer U_t than in layer U_{t-1} and the domain of layer U_t will be of size $\frac{W}{2} \times \frac{H}{2}$. If the stride is three we have a division by nine of the number of units in one channel.

The second sub-sampling operation is called **pooling**. As opposed to augmenting the stride, pooling is not performed directly on the convolution layers but is a layer of its own. Pooling operations simply split the spatial domain of each channel of layer U_{t-1} into windows of size $M \times N$. Only one unit per window is kept on layer U_t after the pooling operation. Depending on the pooling strategy the value of this unit can either be the maximum, the average, or a random value from the units of the window. The bigger the size of the pooling window, the more the spatial domain and the number of units per channel will be reduced in layer U_t.

The last convolution layer is often succeeded by a pooling layer with windows that have the same size as the spatial domain, leaving only one value per channel, to turn the last feature map into a vector of features. This feature vector is then sometimes transformed by a series a fully-connected layers.

Summary Forward propagation To go from pictures to features, Convolutional Neural Networks alternate convolution layers, with varying stride and padding strategies, and pooling layers in a forward propagation as represented on figure 9 page 21. This section defined the operations and the associated weights that are involved in the forward propagation. We will now look into how one can find the optimal values for those weights.

4.3 Training a CNN

The goal of training a Convolutional Neural Network is to learn the best approximation $h_\omega(x)$ of the "perfect" function $h^*(x)$. In order to do so one must find the best set of weights ω. The quality of the approximation is defined by a **loss function** $L(x, h^*(x), h_\omega(x))$ where x is the input data. Intuitively one could say that it measures the distance between $h_\omega(x)$ and $h^*(x)$. Training a CNN consists in finding the smallest possible value of the loss function by optimising ω.

The approximation $h_\omega(x)$ is fully dependent on the set of weights of the model ω. If we assume that we are dealing with a supervised learning problem (see section 2.2 for a definition), $h^*(x)$ can be represented by the association of each training data point x_i with a label y_i (the set of all labels is denoted y). The loss function can thus be re-written $\mathbf{L}(\mathbf{x}, \mathbf{y}, \omega)$ or $L(x, y, h_\omega(x))$.

Training a CNN is thus equivalent to solving an optimisation problem. There exist many techniques to solve optimisation problems. However, they often require that the problem possesses some properties (convexity for example). The minimisation of the loss function with respect to the weights has dreadful properties. Indeed, L depends on ω through h which is a non-smooth, non-convex function of ω. It is clear by looking

at the expression of the activation functions ReLU and Softmax (equations 5 and 6 on page 19) that are composed to get h_ω. The loss function itself $L(x, y, h_\omega(x))$ is also often non-convex. This means that the minimisation of the loss function does not possess the theoretical convergence guarantees associated with linear equation solvers or convex optimisation methods that are used for other machine learning algorithms.

Instead gradient-based iterative optimisers are used to drive the value of the loss function as low as possible, without actually reaching the minimum. They are sensitive to the starting point whereas convex optimisation is very robust in that regard. However, the complexity of the approximation h_ω that a neural network can learn far outweighs this loss of theoretical convergence guarantees.

Gradient descent The loss function is minimised by gradient based iterative methods called **optimisers**. There are many such methods but they are all based on the *gradient descent algorithm*. The gradient descent is designed to reach a local minimum of a differentiable function. It was first introduced by *Cauchy* in 1847. It is based on the well known fact (among mathematicians - the reader is encouraged to look it up if unfamiliar) that the direction (in the space of variables of the function) of the steepest descent (in function value) is given by the negative of the gradient. From that fact one can build the gradient descent iteration which reduces the value of a function. Equation 9 gives the gradient descent iteration for a differentiable function f of variables x, a point x_n on its domain, and a scalar η called the learning rate:

$$x_{n+1} = x_n - \eta \nabla f(x) \tag{9}$$

This is the original version of the gradient descent. Variants that are more robust to the starting parameters, more stable, that do not get stuck in local minimas, etc have been developed and are used in practice to optimise the weights of CNNs. Some of those variants will be introduced in section 4.4. For this section's discussion the original iteration is sufficient.

Equation 10 gives the gradient descent update adapted to the optimisation of the loss function $L(x, y, \omega)$ of a neural network in the parameters ω:

$$\begin{aligned} \omega^{n+1} &= \omega^n - \eta \nabla_\omega L(\omega_n) \\ \text{where } \nabla_\omega L(\omega^n) &= \frac{1}{N} \sum_{i=1}^{N} \nabla_\omega L(x_i, y_i, \omega^n) \\ \text{and } \nabla_\omega L(x_i, y_i, \omega^n) &= \left[\frac{\partial L(x_i, y_i, \omega^n)}{\partial w_1} \dots \frac{\partial L(x_i, y_i, \omega^n)}{\partial w_j} \dots \frac{\partial L(x_i, y_i, \omega^n)}{\partial w_M} \right]^T \end{aligned} \tag{10}$$

with (x_i, y_i) pairs of training data points and labels, N the total number of training data pairs, and M the total number of parameters.

Backpropagation Computing the gradient $\nabla_\omega L(x_i, y_i, \omega^n)$ is not a trivial task. A standard numerical procedure to compute gradients is the numerical differentiation. It computes partial derivatives according to equations similar to equation 11:

$$\frac{\partial L(x_i, y_i, \omega)}{\partial w_i} = \frac{L(.., \omega + \epsilon) - L(..., \omega)}{\epsilon} \tag{11}$$

It involves re-evaluating M (the number of weights) times the network for all N training points at every gradient iteration. Because the networks have millions of parameters, and the training sets usually have ten of thousands of points, it can quickly result in billions of forward propagations which is a numerically infeasible task.

The preferred approach to evaluate the gradient in a neural network is the **backpropagation** algorithm. It was first introduced by *David E.Rumelhart et-al* in 1986 [26] and it is not computationally expensive. The backpropagation starts from the value of the loss $L(x_i, y_i, \omega^n)$ that is obtained at the end of the forward propagation. It then propagates the loss information back to each weight by going backwards into the graph. Figure 11 shows an example for a MLP.

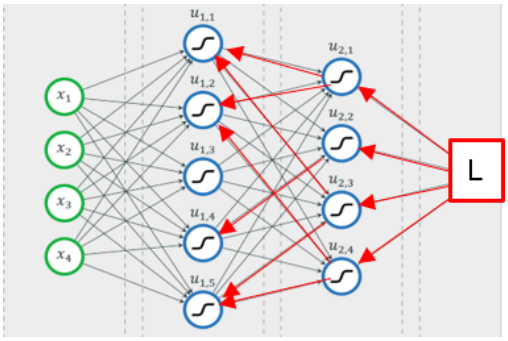


Figure 11: Backpropagation through a MLP (modified from [22] with permission)

The dependence of the loss function on each weight w_i is computed by recursively applying the **chain rule** which is the formula to compute the derivatives of a composite function. It is due to *Leibniz* (1676). We stated in section 4.1 that the output of a network is the result of the (structured) composition of all the activation functions of the neurons. The chain rule is thus perfectly adequate. Equation 12 shows how the chain rule is applied in a neural network:

$$\frac{\partial L}{\partial u_i} = \sum_{u_j \in \text{output } u_i} \frac{\partial L}{\partial u_j} \frac{\partial u_j}{\partial u_i} \tag{12}$$

Because the backpropagation goes from right to left, $\frac{\partial L}{\partial u_j}$ is known at the time of the computation of $\frac{\partial L}{\partial u_i}$. The backpropagation made the minimisation of the loss function possible for large scale neural networks such as CNNs.

4.4 Training pitfalls and solutions

Section 4.3 mentioned that the gradient-based optimisers have no convergence guarantees when dealing with neural networks. For the minimisation of the loss function to happen relatively smoothly, the deep learning engineer must actually avoid multiple pitfalls. There are a variety of techniques that have been developed to overcome them. This section discusses the most common pitfalls one can fall into when training a neural network and describes solutions to overcome them.

4.4.1 Prevent vanishing gradients

The optimisers are all based on the gradient descent (equation 9 on page 24). This method iteratively decreases the value of the loss function towards a local minimum. But if the gradient becomes very small the descent will stop. This behaviour happens particularly often in neural networks and is called **vanishing gradients**.

The deeper a network gets, the most likely the vanishing gradient phenomenon is likely to occur. Indeed, in a very deep network the loss function depends on the weights in the "early" layers (the ones close to the inputs of the network) through many intermediate units. There are thus many intermediate partial derivatives involved in the computation by chain rule (equation 12) of the partial derivative of the loss with respect to those early weights.

The value of each of those intermediate partial derivatives depends on the corresponding weight and on the activation function. Equation 13 gives a generic example of the dependence:

$$\begin{aligned} u_i &= f(z) \quad \text{with } z = w_1 u_{i-1} + w_2 ... + ... \\ \frac{\partial u_i}{\partial u_{i-1}} &= \frac{df}{dz} w_1 \end{aligned} \tag{13}$$

Some activation functions make those derivatives very small. Indeed, they are almost constant on the most part of their domain, hence making $\frac{df}{dz}$ almost zero on that portion of the domain. The tanh function $\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$ is one such function. The intermediate partial derivatives of units that use those functions can quickly become almost zero. When there are many such partial derivatives involved in a chain rule computation, the likelihood that the result is almost zero becomes important. This is one of the reasons why the gradient can vanish. A solution is to use different functions, such as the ReLU functions (equation 5 page 19) for which $\frac{df}{dz} = 1$ on half their domain.

Yet this is not enough to completely overcome vanishing gradient. Specific network architectures, the **Residual Networks**, also directly combat the vanishing gradient phenomenon, enabling neural networks to go much deeper than previous architectures. Residual networks will be discussed in chapter 5.

4.4.2 Improve gradient descent iteration

The gradient descent algorithm has no convergence guarantee in neural networks, as explained in section 4.3. However, there are a few modifications one can make to the algorithm that can help with the convergence.

Mini-Batch gradient descent The first one is to use a mini-batch version of the gradient descent. The update rule of the traditional gradient descent algorithm is defined in equation 10 page 24. In that traditional update the gradient is computed by averaging over all the data of the training set. The mini-batch version computes the gradient by averaging over a smaller number of data points called a **batch**. The update rule of the weights stays the same but the gradient becomes

$$\nabla_\omega L(\omega^n) = \frac{1}{B} \sum_{i=j}^{j+B-1} \nabla_\omega L(x_i, y_i, \omega^n) \tag{14}$$

where B is the size of the batch. The weights are thus updated more often, without using the N data points. The number of iterations needed for a model to use all N points to update its weights is called an **epoch**. The length of the optimisation procedure is often defined in number of epochs.

The batch size B has an influence on the convergence of the descent. The smaller the batch the noisier the descent will be between each iteration. This noisy character is useful up to a certain point because it helps escape local minima. But if the gradient becomes too noisy there will be no convergence at all. It is thus important to choose a good batch size to craft the noisiness. There are no rules to set it but most training procedures use a batch size between 36 and 128.

Learning Rate Decay Similarly one can also tune the learning rate of the gradient descent during training. The learning rate determines the size of the step taken at every iteration in the direction of the local steepest decrease (in loss function value) given by the gradient.

Decaying the learning rate as the number of epoch grows is a very common strategy. During the first part of training it is useful to have a relatively big learning rate to move the weights from their initial random values towards a better region of the space of weights. It is important to note that the initial value of the weights is also important. Generally one starts with small random values but other techniques are possible (see section on transfer learning).

Once the weights have reached that "good" region the learning rate must decrease. Indeed, if it stays too big the descent will oscillate between good but sub-optimal sets of weights. It is thus very important to decay it as the training progresses to ensure that the model to descent towards a better, locally optimal, solution.

Momentum Another strategy to improve convergence is to modify the update rule of the gradient descent. One way to do so is to use the momentum update strategy (*Polyak,* [27]). The goal is to average out the gradient direction over time. To do so the update rule uses previous gradients as well as the current gradient to compute the *momentum* to apply to the weights. The momentum is the vector of the displacement in the parameter space. There is a clear analogy with the physical notion of momentum. Mathematically the momentum update is

$$\begin{aligned} v^{n+1} &= \mu v^n - \eta \nabla_\omega L(\omega^n) \\ \omega^{n+1} &= \omega^n + v^{n+1} \end{aligned} \tag{15}$$

where the vector v is the momentum (it is actually v for velocity with unit mass assumed). The parameter μ decides "how quickly the contribution of previous gradient exponentially decays" (*Goodfellow,* [5]).

Optimisers Many optimisers, i.e gradient descent based algorithms, use a combination of mini-batch update, learning rate decay and momentum to create their own update rule. We can cite *SGD*, *RMSprop*, *AdaDelta*, and *Adam* as the most commonly used. There is no global consensus on which one to use in a given situation. *Schaul et al.* [28] presented a comparison of the optimisers across a wide variety of learning situations but no single best algorithm has emerged.

4.4.3 Combat overfitting

Another problem that often occurs when training neural networks is that the weights learned from the training set do not generalise well to other data points from the same distribution. This phenomenon is called *overfitting* and was already introduced in chapter 3. Formally, "a model overfits the training data when it describes features that arise from noise or variance in the data, rather than the underlying distribution from which the data were drawn" (*Geoffrey I. Webb,* [29]). The best way to combat overfitting is to use more training data. However, it is often unfeasible because of a lack of data available or due to computing or memory capacities limitations. There are multiple other methods that help combat overfitting in neural networks. We will describe the most commonly used.

Regularisation Regularisation originates from *Occam's Razor* that is often formulated as "of two hypotheses H and H', both of which explain E, the simpler is to be preferred" (*I. J. Good,* [30]). Regularisation techniques try to maintain the neural network as "simple" as possible, i.e to limit its complexity, by modifying the loss function. The complexity of the model can be measured in many ways. One of the most common ones in neural networks is to use the sum of the L_2 norms of the weights. This leads to

the L_2 regularisation where the loss $L(\omega)$ becomes

$$L'(\omega) = L(\omega) + \lambda \sum_{j \in J} ||w_j||_2^2 \tag{16}$$

where λ is a scalar parameter and J is a subset of the weights.

Data Augmentation Data augmentation is a way to implement the best solution to overfitting, get more data, without actually sourcing new data. The idea is to add random noise to the training data so that the training algorithm will see a different version of the same data at every epoch. As an example, when the training data is a set of pictures, one can randomly crop the pictures to rectangles three quarters the size of the original picture. And if the position of the rectangle in the picture varies at every iteration the training algorithm will see a slightly different version of that picture every time.

One can also shuffle the order in which the data is fed to the algorithm at every epoch. It is not formally data augmentation but it is a crucial technique to avoid overfitting and is thus worth mentioning.

Dropout Dropout, introduced by *G. E. Hinton et al* [31] is a technique inspired by a procedure called *bagging*. Bagging consists in training multiple networks on the same training set and then averaging their predictions to create a single model. However, it is highly impractical to train many neural networks to perform the same task. Dropout enables to perform bagging with a very large number of networks originating from the same network with almost no computational cost. The idea is to randomly switch off neurons in the network, thereby creating a new network at every training iteration. The procedure is illustrated in figure 12.

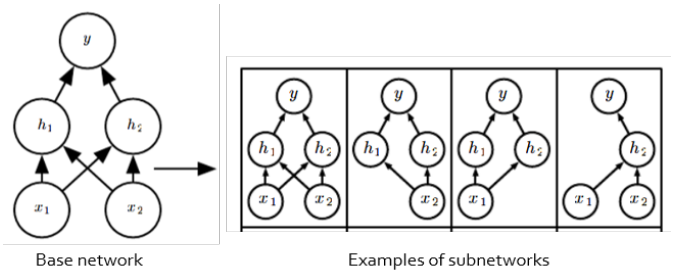


Figure 12: The Dropout procedure [5]

The final network is a "mean version" of the base network where all units are used but where their outputs are divided by the probability of them being switched off. In big networks such as CNNs, dropout is applied per layer. Each neuron of that layer can then randomly be switched off with a user-defined probability during training.

Dropout greatly reduces overfitting because it "prevents complex co-adaptations on the training data" (*G. E. Hinton et al*, [31]). Indeed, hidden units (in the same layer) can not rely on each other to learn features as some of them are randomly switched off at every training iteration. This pushes them towards learning simpler features from the distribution and not complex features from the training data. This characteristic explains why dropout is viewed as a regularisation technique (the definition of regularisation is given above).

Batch Normalisation Batch normalisation is a technique that was initially created to prevent the *Internal Covariate Shift*. It is the fact that "each layer's input distribution changes during training, as the parameters of the previous layers change" (*Sergey Ioffe*, [32]). It is problematic because if at iteration i the inputs of layer l are grouped in a part of the domain, the layer l will learn a function that approximates the true distribution very well on that part of the domain but maybe not on the rest of the domain. For points in another part of the domain the approximation will be poor and the loss big. Globally this means that the network will learn a good approximation of the true distribution on the part of the domain from which the training points are sampled. But if the testing distribution is drawn from a different part of the domain the network will not generalise well.

The solution introduced by *Sergey Ioffe*, [32] is to normalise the inputs of the layers by using the mini-batch statistics to ensure that the each layer's input distribution has a mean of 0 and a variance of 1. This is similar to the *whitening* procedure in image processing. When applied to a linear layer (weights γ and bias β) it yields the algorithm in figure 13

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Figure 13: Batch Normalisation algorithm [32]

Batch normalisation is an effective training procedure. But it turns out that it does not reduces Internal Coveriate Shift as it intends to do as shown by *Shibani Santurkar et-al* [33]. Nevertheless it is undeniable that it reduces overfitting even if how it does it

remains mysterious. It is thus widely used in practice.

4.4.4 Use transfer learning

Convolutional Neural Networks have a huge number of weights, very often in the tens of millions. When one trains a model from scratch it is common to initialise the weights with small values as mentioned earlier. When there are many weights training the network so that they all reach a good value requires a lot of time and computing resources.

Another strategy is to not train the model from scratch but to re-use weights from a similar model that have been learned on another dataset and/or for another but related task. This is called transfer learning and the resulting model is said to be **pre-trained**. The idea is that if a model has been trained to recognise pictures of cars, the features of the inner layers are relevant to recognise pictures of bikes. If one wants to train a network to recognise bikes he can thus re-use the weights of the network trained on cars. He can then suppress the last car specific layers, design new bike specific layers and train only those layers to recognise bikes instead of cars. The optimiser only updates the weights of the re-designed layers in that case. The others are *frozen*. Using transfer learning makes it possible to obtain in hours and with a a few thousand images performances that could take weeks and hundreds of thousands of images otherwise.

If one possesses enough data, he can also decide to use transfer learning as described above but also to update the weights of the pre-trained layers. This is called **fine-tuning** and will yield even better performances. However, in that case one should start by training the new, re-designed, layers only for a few epochs with the pre-trained layers frozen so that the randomness generated by the initialisation of the new layers does not destroy the weights of the pre-trained layers. This first part of the training is called the *warmup*.

Transfer learning makes it possible to reach state of the art performances with limited time and computing resources and is thus a very powerful technique.

Solutions but no magic recipe

This section outlined some of the most common problems that can occur while training a neural network. It also introduced various techniques to solve or at least reduce those problems. None of those techniques provide convergence guarantees to the minimisation of the loss function. Furthermore most of them require to set some **hyperparameters** (the size of the batch, the dropout probability, etc) and there are no guidelines to set those parameters. However, using the techniques discussed in this chapter is an absolute necessity to reach state of the art performances on all problems solved with deep learning.

Summary

The intended goal of this chapter was to provide a practical introduction to the way deep learning models are built to solve recognition problems. It first defined a neural network as an organised composition of functions. The function resulting from the composition can be very complex and has the capacity to approximate any "perfect" function that would solve a given recognition problem. Then it focused on the mechanics of Convolutional Neural Networks, the models specific to image processing tasks. It defined how those networks transform a picture into a feature vector by a process called the forward propagation which involves a lot of weights. It then described how gradient descent based algorithms can be used to obtain an optimal set of weights as defined by the minimisation of a loss function. It introduced the backpropagation algorithm that computes the gradient efficiently in a neural network. Finally it looked at the main issues, and the related solutions, that come with using gradient descent to optimise neural networks.

Chapter 5

Models

5.1 Objectives of a model for Re-Identification

The primary goal of this thesis is to build a model that can answer the Pair Re-Identification question : *Is this pair of images from the same identity ?* This question belongs to the broad field of recognition. Chapter 2 explained that the preferred method for solving this class of problems is to build deep learning models. As a matter of fact state of the art performances for the traditional Re-ID task (see definition in the Introduction) are obtained with the use of deep learning models.

The Pair Re-ID question is not directly addressed in the literature and there is thus no best approach for building deep learning models that can answer it. However, as for all recognition problems the solution should be based on learning a feature space. The characteristics of that feature space depend on the problem at hand. Yet the fact that the feature space possesses some characteristic is rarely the ultimate goal of the model. It is the case because it is the best way to solve a given task, classification for example.

To answer the Pair Re-ID question a deep learning model needs a measure of the similarity of the two images of the pair. The euclidean distance in the feature space learned by the model between the embeddings (feature vectors) of the two pictures is one such measure. The characteristics of the feature space relating to its distance are thus prevalent. The objective of models for Pair Re-ID is formulated in that regard:

Definition 1 (Pair Re-ID objective)**.** *The model learns a feature space where pairs of embeddings of the same identity are separated by a smaller distance than pairs of embeddings of different identities.*

Because Pair Re-ID is related to traditional Re-ID we will use models that have been developed for the latter, eventually modify them, and evaluate them on the Pair Re-ID objective. The performances of the models for Traditional Re-ID will also be analysed for comparison with state of the art models and to compare and contrast both types of Re-ID. It is thus useful to formulate the Traditional Re-ID objective in terms of distance between embeddings in a feature space:

Definition 2 (Traditional Re-ID objective)**.** *The model learns a feature space where the n closest embeddings to a query image of a given identity are from the same identity.*

The Pair Re-ID and Traditional Re-ID objective are related in that they should both learn a feature space where the distance between pairs of the same ID should be small. However, for Traditional Re-ID it is sufficient that the distances between the few closest pairs of the same ID are small. For Pair Re-ID those distances should be small for all

pairs. Moreover and above all the distance between pairs of different IDs should be bigger than the distance between all pairs of the same ID. This should not specifically be the case for Traditional Re-ID.

This chapter presents the models implemented for the purposes of this thesis to meet the Pair Re-ID objective. The models differ mostly in their supervision. This chapter thus puts an emphasis on describing the loss function that is used to train each model. The architectural differences between the models are mainly adaptations to feed appropriate inputs to the different loss functions. This chapter uses many of the tools introduced in chapter 4.

5.2 ResNet-50, the backbone

All the models for Re-ID have the same two-part architecture. The first part is a series of layers from a Convolutional Neural Network and is called the **backbone** of the model. The second part is a series of fully connected layers and is called the **head** of the model. The backbone of all Re-ID models presented in this thesis comes from a Convolutional Neural Network called **ResNet-50**. This CNN belongs to the family of networks called *residual networks*. The idea for their architecture was first introduced by *Kaiming He et al* in 2015 [34].

The motivation was to create an architecture that could be very deep (hundreds of layers) without succumbing to the problem of vanishing gradients that destroys the convergence of the gradient descent (as explained in section 4.4). It is recalled that the deeper the network, the heavier the vanishing gradient phenomenon. On the other hand it has been shown that the deeper the network, the better the feature space it learns will generalise outside the training set. There is thus a big incentive to build networks that can go deep.

The ResNet architecture directly combats the vanishing gradient phenomenon by introducing the **skip connection**. The idea is very simple. The architecture of a CNN is usually a sequence of convolution layers and pooling layers. This architecture is purely sequential, each layer depending on all the preceding ones. The idea of residual networks was to insert some parallelism into the architecture. There are thus, for several portions of the network, parallel paths to the main sequential path that *skip* this main path. Those parallel paths sum up at the end of the portion. If the gradients in either the main path or the parallel path in that portion become very small, the gradients beyond the summation point will not be impacted. The parallel path is also much simpler than the main path, involving less units, hence reducing the opportunity for vanishing gradients.

Two types of blocks of layers implement the skip connection: the **identity block** and the **convolution block**. The identity block is used when the input and output feature maps of a block should have the same dimensions. It contains convolution layers with batch norm and ReLU activation (see chapter 4 for the definition of those concepts).

The skip connection simply sums the input of the block with the output computed by the main path. Figure 14 shows the connection.

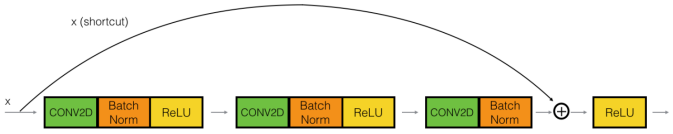


Figure 14: An Identity Block [35]

As explained in section 4.2, CNNs gradually transform a picture into an embedding by regularly reducing the width and height of the feature maps and augmenting the number of channels. The identity block does neither of those things. There is thus the need for a second block, called the convolution block. Its main path also contains convolution layers with batch norm and ReLU activation. But the skip path also has a convolution layer of which the number of kernels, stride and padding parameters (see chapter 4 for definitions of those concepts) are set up to resize the feature map to the desired dimension. The main path naturally also reduces the feature maps to the same dimensions. Figure 15 shows a convolution block.

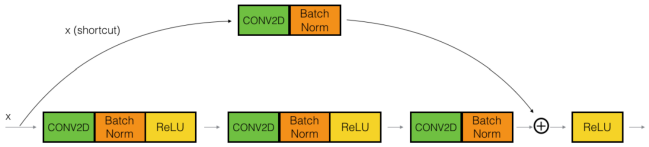


Figure 15: A Convolution Block [35]

Resnet-50 is composed of four stages of identity blocks and convolution blocks with varying kernel sizes, number of kernels, strides and paddings parameters. There is a first stage, composed of a convolution layer and a pooling layer, that precedes those four. The last part of the CNN is a block of fully-connected layers called the head of the network. The global structure of the ResNet-50 and the values of the aforementioned parameters are given in appendix A.

When ResNet-50 is used as a backbone in the two-part architecture described at the beginning of the section, the head is removed and replaced by a new one specifically designed for Re-ID. The backbone of all the models presented in the following sections is a ResNet-50 whose head is removed. New heads, specific to each model, are designed. They differ mainly to better fit the supervision associated with each model.

5.3 Classification

The section will present a model for Re-ID whose training is supervised by classification. It uses the two-part architecture described in section 5.2. The backbone of the

model is thus a ResNet50 whose head has been removed. The new head is constructed so that the model can be trained for classification. This model will thus be called the *Classification (CL)* model. Supervising a model for Pair Re-ID with classification may seem unappropriated as neither the Traditional Re-ID task nor the Pair Re-ID question are classification problems (classification problems are defined in chapter 2).

However, in order to perform classification deep learning models first learn a feature space like for any other recognition problem. To classify a picture they first map it to that feature space by forward propagation and then use the obtained embeddings to perform classification. A feature space that offers a strong basis to classify persons identities should also perform well for both the Pair Re-ID and traditional Re-ID objectives. The approach of training the model for classification and then using the learned feature space to solve the Re-ID task actually turns out to yield very good performances for Traditional Re-ID. It is now a mainstream approach in the field [36]. *Zhedong Zheng et-al* [37] were the first to offer a strong discussion of the approach.

5.3.1 Cross-Entropy

To train a model for classification the last layer of the head should have as many units as there are classes, in our case as many as there are identities in the training set (N_{ID}). Each unit corresponding to one identity. The units of this last layer should have a *softmax activation* (introduced in equation 6 page 19):

$$u_i = \text{softmax}_i = \frac{\exp z_i}{\sum_{j=1}^{m} \exp z_j}$$

In that case the output represents a probability distribution over the N_{ID} classes. The classification label that the model assigns to an image is the maximum over the probabilities.

Those probabilities are directly used in a loss function called the Cross-Entropy. Its goal is to optimise the model for classification. Mathematically it tries to maximise $p(h_w(x_i) = y_i|x_i)$, the probability that the model outputs the correct label, $h_w(x_i)$ being the function corresponding to the forward propagation through the network, y_i being the label, and x_i the input. In our case the output $h_w(x_i)$ corresponds to the output of the last layer before it is passed to the softmax activation. The probability that the model outputs the correct label among all labels c_j is thus:

$$\begin{aligned} p(h_w(x_i) = y_i|x_i) &= \prod_{j=1}^{N_{ID}} p(h_w(x_i) = c_j|x_i)^{[y_i=c_j]} \\ \text{where } [y_i = c_j] &= \begin{cases} 1 & \text{if } c_j = y_i \\ 0 & \text{otherwise} \end{cases} \\ \text{and } p(h_w(x_i) = c_j|x_i) &= \text{softmax}_{c_j}(h_w(x_i)) \end{aligned} \tag{17}$$

Maximising this function for the set of parameters ω of the network is very similar to a maximum likelihood analysis which is used in statistics to optimise the parameters of a probability distribution. It is well known that minimising the negative of the log-likelihood in such a case is equivalent and much easier. The Cross Entropy loss function is thus obtained by taking the negative log of $p(h_w(x_i) = y_i|x_i)$ and by averaging for all input x_i of the training batch (of size B - see section 4.4 for a definition of mini-batch training).

$$L_{CL} = -\frac{1}{B}\sum_{i=1}^{B}\sum_{j=1}^{N_{ID}} [y_i = c_j] \ln\left(p(h_w(x_i) = c_j|x_i)\right) \tag{18}$$

5.3.2 Architecture

To go from the three dimensional feature maps obtained at the end of the backbone to the last softmax layer, one needs to add at least two intermediate layers. The first one performs a *Global Average Pooling* operation which reduces the features maps of size $(h, w, 2048)$ to a fully connected layer of size 2048. The second one is a linear layer to transform the 2048 units in a softmax layer with N_{ID} units. We noticed that adding a third intermediate linear layer (with batch normalisation) with less units than the softmax layer resulted in better results on the validation set, and thus to better generalisation. This new layer is inspired by the architecture developed by *Hao Luo et-al* [38]. The resulting head architecture is represented in figure 16.

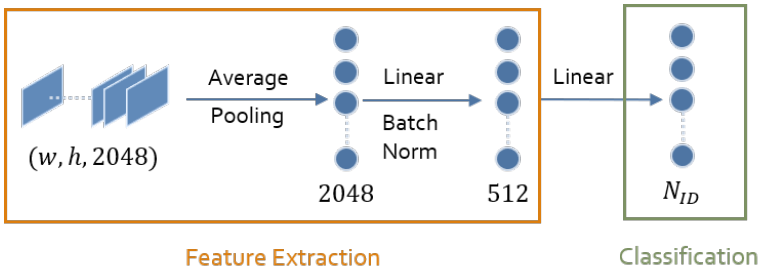


Figure 16: The head of the Classification Model

The last softmax layer is of no use for Re-ID. Indeed, when given an input picture the model shouldn't output a probability distribution of the image belonging to an identity but an embedding. When used for Re-ID the last "classification" layer is thus dropped and the resulting network works as a feature extractor.

Last Stride Parameter According to *Hao Luo et-al* [38] "Higher spatial resolution always enriches the granularity of feature." This statement means in our case that bigger feature maps at the end of the ResNet-50, i.e higher spatial resolution, would produce better embeddings. To produce bigger feature maps they proposed to reduce the stride

of the last layer of the backbone from 2 to 1. This makes the size of the feature maps, when presented with (256,128,3) sized input pictures, go from $(8, 4, 2048)$ to $(16, 8, 2048)$.

5.3.3 Label Smoothing

The method of learning a feature space based on classification and Cross Entropy has proven effective for Traditional Re-ID. However, it has an inherent drawback. Indeed, the formulation of the Cross Entropy loss is such that the model will maximise the confidence in all the labels it predicts [36]. The network tries to separate images of different IDs that look a lot alike, such as two bald white men in suits, as much as images of different IDs that look nothing alike, such as one of these men in his suit and a twelve year old girl in a pink dress. This leads the network to struggle with identities that are very similar for traditional Re-ID.

George Adaimi et-al [36] tackled this drawback. They proposed three methods to combat the issue. This thesis has implemented one, deemed as the simpler and most effective: the modification of the Cross Entropy with label smoothing. Label smoothing reduces the confidence in the model's predictions by modifying the ground truth distribution.

$$g := [y_i = c_j] = \begin{cases} 1 & \text{if } c_j = y_i \\ 0 & \text{otherwise} \end{cases} \longrightarrow \quad g_{LS} := [y_i = c_j] = \begin{cases} 1 - \epsilon & \text{if } c_j = y_i \\ \frac{\epsilon}{N_{ID}-1} & \text{otherwise} \end{cases} \tag{19}$$

This method enables the network to share the representation of an identity between multiple classes which enables to better deal with persons identities that are truly similar. *George Adaimi et-al* [36] showed that the method was effective for Traditional Re-ID.

5.4 Metric Learning

The second model also follows the two-part architecture described in section 5.2. Whereas the head of the model described in section 5.3 was designed to be trained for classification this one is designed to directly tackle the learning of a feature space in which the distances have some desired properties. This is the definition of metric learning as introduced in section 2.2. This model will thus be called the *Metric Learning (ML)* model.

Researchers [39] [40] have developed a family of loss functions that implement a metric learning supervision for Traditional Re-ID. These loss functions are called **triplet losses**. Models trained with those losses have shown up to be less efficient on Traditional Re-ID tasks than models supervised with classification, and are not really considered state of the art anymore. Nevertheless we will show that those losses fit the Pair Re-ID objective very well which make them particularly interesting for this thesis.

5.4.1 Triplet Loss

The Triplet Loss supervises training so that pairs of the same identity are separated by a smaller distance than pairs of different identities. The reader can notice that this is exactly the Pair Re-ID objective. A model supervised by those losses should thus fulfil the Pair Re-ID objective very well, better than it fulfils the Traditional Re-ID objective.

To enforce that distances between embeddings of the same identity should be larger than distances between embeddings of different identities, the Triplet Loss takes triplets of embeddings as input. Each triplet is based on one embedding called the *anchor* e_a, and selects the other two according to the anchor. The second one e_p is an embedding of the same identity as the anchor (called positive embedding) and the third one e_n, of a different identity (called negative embedding). The Triplet Loss enforces the constraint on the distances between those embeddings as in equation 20:

$$\begin{aligned} L_{ML} &= \frac{1}{B}\sum_{a=1}^{B}(m + ||e_a - e_p||_2^2 - ||e_a - e_n||_2^2)_+ \\ \text{where } e_a &= h_w(x_a), e_n = h_w(x_n), e_p = h_w(x_p) \\ \text{and } [m+a]_+ &= \max(0, m+a) \text{ is the Hinge function} \end{aligned} \tag{20}$$

where m is a parameter that sets the desired margin between the same ID distances and different ID distances, and B is the size of the batch. Because the goal of the training is to minimise the loss, the network will learn an embedding that makes $||e_a - e_n||_2^2$, the different ID distance, as big as possible and $||e_a - e_p||_2^2$, the same ID distance, as small as possible in order for the two distances to be separated by at least m. One can see that it is exactly the Pair Re-ID objective.

Soft Margin The role of the Hinge function $\max(0, m+a)$ is to avoid correcting already correct triplets, i.e triplets where the different ID distance is already bigger by m than the same ID distance. However, for Re-ID it can be useful to ensure that the separation is as big as possible. *Alexander Hermans et-al* [41] proposed using for that purpose a smooth version of the Hinge function called the soft margin Hinge function (sometimes denoted *log1p* in the computing libraries):

$$(m+a)_+ = \ln\left(m + \exp\left(a\right)\right) \tag{21}$$

Batch Hard Triplets We explained that each input triplet for the loss is based on one embedding called the anchor and that the two other members of the triplet are selected according to that anchor. But how does one build a triplet from the training set ? There are multiple strategies and the fact that there is no easy answer to this question is one of the main drawbacks of Triplet Loss based trainings.

The first intuitive solution is to consider all possible triplets leading to an effective training set size of N^3 which is highly impractical because of the high need in computing

resources. Furthermore, after a few epochs, an important portion of those triplets would become useless as the same ID distance would become much smaller than the different ID distance. Those triplets are deemed too easy. In opposition, there exist triplet building methods that mine *hard triplets*. Those are triplets where the same ID distance and different ID distance are close. There are many different rules for mining hard triplets. Some of them are also very greedy in computing power and time and need to be carried out *offline*, before the actual training takes place.

For the purposes of this thesis, the implemented strategy for mining hard triplets is such that the triplets are batch hard triplets. It was introduced by *Alexander Hermans et-al* [41]. It can be performed online (during training). It is based on selecting the hardest positive and negative embeddings for each anchor **within its batch**. All points are considered as anchors, there are thus N triplets per epoch.

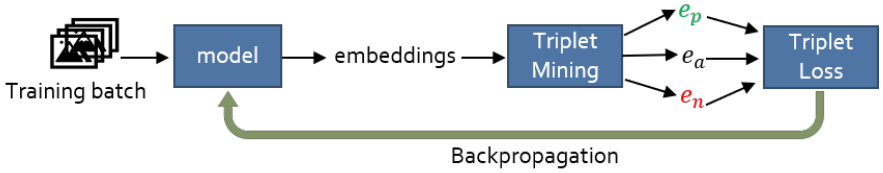


Figure 17: Training procedure for the Metric Learning model

Hardest for the positive embedding means that, among the other points of the same ID as the anchor in the batch, the one with the the biggest distance to the anchor is selected. For the negative embedding, it is the point of a different ID with the smallest distance among the points in the batch. To ensure that the different batches offer a consistent choice for triplets they are all formed of P IDs and K samples per ID. There are thus $P \times K$ data points per batch. The selection of the hardest points within the batch makes it possible for the selection to be performed during training. If the hardest points within the whole training set were selected it would be impossible.

5.4.2 Architecture

The head of the model is designed to work as a feature extractor. It is made up of a global average pooling layer followed by two linear layers. The first one uses a ReLU activation and batch normalisation. The resulting architecture is represented in figure 18.

This architecture is inspired by *Alexander Hermans et-al* [41]. The last stride parameter is also decreased to 1 to obtain bigger feature maps and thus better embeddings as for the Classification model (see section 5.3).

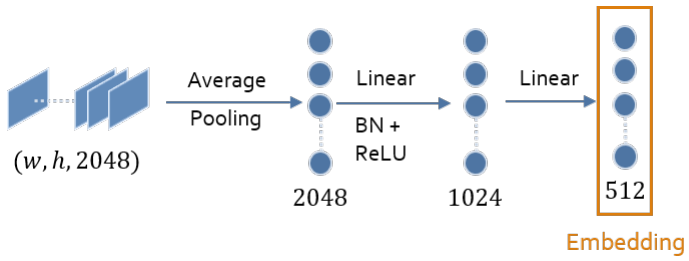


Figure 18: The head of the Metric Learning Model

5.5 Classification and Metric Learning, two incompatible approaches ?

Two models for Re-Identification have been introduced at this point. Both learn a feature space but one is trained for classification and the other is trained for metric learning. The question that naturally comes up is: which one to use for Pair Re-ID ? On one hand, the formulation of the metric learning loss (the Triplet Loss) better fits the Pair Re-ID objective. On the other, it is proven that the embeddings learned through classification are better for Traditional Re-ID which is a closely related problem.

But what if we didn't need to choose ? The remainder of this chapter will introduce two models that combine both approaches. The goal is to benefit from the features obtained by training a model for classification that have proven more effective for Traditional Re-ID, and from the formulation of the Metric Learning training that closely fits the Pair Re-ID objective (more than it does fit Traditional Re-ID).

5.6 Successive Classification and Metric Learning

The first model that combines classification and metric learning uses both training objectives successively. The idea is to first learn a feature space with a classification based training and then, with a second training, to learn a feature space by metric learning starting from the features learned for classification.

This model is actually composed of two inner models, one corresponding to each of the two trainings. The first is a CNN which learns a first feature space from the pictures with classification supervision. It will be called the **inner classification model**. The second model can have different structures but learns a second feature space with metric learning supervision from the embeddings in the first feature space. It will be called the **inner metric learning model**. The hope is that the embeddings in the second feature space will be better suited for Pair Re-ID than the ones in the first feature space. The model resulting of the succession of the two inner models will be called the CL2ML (for classification to metric learning) model.

The first training trains an inner classification model that is exactly the same as the CL model described in section 5.3. It transforms the training set pictures into embeddings in the first feature space. Those embeddings compose the training set for the inner metric learning model. We have designed two different models to learn the second feature space through metric learning. They both use a variation of the Triplet Loss (equation 20 page 39) which, as explained in section 5.4, exactly fits the Pair Re-ID objective.

5.6.1 Multi-Layer Perceptron

The first inner metric learning model is a Multi-layer perceptron (MLP): a neural network composed of fully connected layers only (see section 4.1). This network is very similar to the head of the Metric Learning model which is represented in figure 18 on page 41. This is natural because both structures are composed of fully connected layers and are designed to learn a feature space with metric learning supervision. Yet the fact that the MLP starts from the embeddings produced by the inner classification model resulted in a few modifications in the architecture. The regularisation by batch normalisation is replaced by a Dropout regularisation (with probability 0.4) and the dimension of the final embedding is 256 and not 512. The architecture is represented in figure 19. Those changes are motivated by the monitoring of the loss on the validation set and have no theoretical justification. The MLP is trained with the Triplet Loss and batch hard triplet mining just like the ML model.

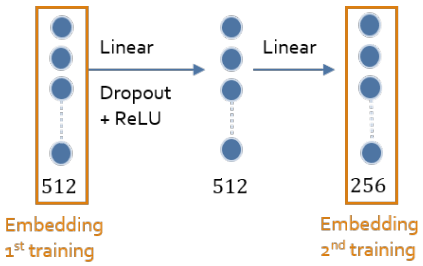


Figure 19: Multi-Layer Perceptron for Metric Learning

5.6.2 Mahalanobis Matrix

The second inner metric learning model learns a *Mahalanobis distance*. A Mahalanobis distance is associated with a positive semi-definite matrix M ($M \succcurlyeq 0$) and is defined as in equation 22:

$$D(x, y) = \sqrt{(x - y)^T M (x - y)} = ||x - y||_M \tag{22}$$

When one speaks of learning a Mahalanobis distance, one means learning the entries of the matrix M. The model is thus the matrix M hence it is not a deep learning model

but a machine learning model. It is the "traditional" type of model implemented for Metric Learning [42].

Metric learning was defined as training a model with the objective of explicitly learning a feature space. It is not obvious from equation 22 that the Mahalanobis based model learns a feature space. Yet we can rewrite equation 22 so that it becomes very clear. We start from the fact that any positive semi-definite matrix can be decomposed as: $M = L^T L$. Using this decomposition of M and with a little linear algebra we obtain the equivalent expression of the Mahalanobis distance given by equation 23:

$$D(x, y) = \sqrt{(Lx - Ly)^T (Lx - Ly)} = ||x - y||_M \tag{23}$$

It is here obvious that the vectors x and y are projected into a new feature space by L.

Mahalanobis matrices can be trained so that the corresponding distance has any given property. The property one wishes to obtain from the distance depends on the design of the loss function that supervises the training. We've seen that the Triplet Loss corresponds directly to the Pair Re-ID objective (in section 5.4). We will thus use a version of the Triplet Loss adapted to the training of a Mahalanobis distance. This loss was introduced by *Weibin Li et-al* [43]. Its expression is given by equation 24:

$$\begin{aligned} L_{MOML} &= \frac{1}{B} \sum_{a=1}^{B} \frac{1}{2} ||M_t - M_{t-1}||_F^2 + \gamma [1 + Tr(M_t A_a)]_+ \\ \text{with } A_a &= (e_a - e_p)(e_a - e_p)^T - (e_a - e_n)(e_a - e_n)^T \end{aligned} \tag{24}$$

where the triplet (e_a, e_p, e_n) is a triplet of embeddings in the first feature space. The triplets for this training are also batch hard triplets (see section 5.4). The index t corresponds to the training iteration (one training iteration is one batch of data). The term $||M_t - M_{t-1}||_F^2$ is a regularisation term ($||\cdot||_F$ is the Frobenius norm). The other term $[1 + Tr(M_t A_a)]_+$ is similar to the Triplet Loss for the metric learning model (equation 20 page 39):

$$[1 + Tr(M_t A_a)]_+ = \max(0, 1 + ||e_a - e_p||_M - ||e_a - e_n||_M) \tag{25}$$

The minimisation of L_{MOML}, with $M \succcurlyeq 0$, is a convex problem that *Weibin Li et-al* [43] defined as the *Mahalanobis Online Metric Learning (MOML)* problem. The minimisation is carried out by a gradient descent based optimiser just as for neural networks.

5.7 Simultaneous Classification and Metric Learning

The second approach that combines classification and metric learning supervisions to learn better features for Pair Re-ID uses both objectives simultaneously. A single model is thus trained and it learns a feature space that is influenced by both a classification

objective and a metric learning objective. The idea of combining both losses during the same training and the resulting architecture is inspired by *Hao Luo et-al* [38]. We will call this model the CL+ML model.

The model is a CNN that follows the two-part architecture introduced in section 5.2. Its backbone comes from a ResNet-50 and it has a head composed of a pooling operation succeeded by fully connected layers. It is therefore similar to the models CL and ML introduced in sections 5.3 and 5.4 respectively.

5.7.1 Triplet Loss and Cross Entropy

The model is supervised by a loss function that combines both classification and metric learning. This new loss was built by simply adding up the loss functions for classification and metric learning (equations 18 page 37 and 20 page 39 respectively):

$$L_{CL+ML} = \alpha L_{CL} + \beta L_{ML} \tag{26}$$

where α and β are scalar parameters.

The two losses do not require the same type of input from the model. Indeed, we explained in section 5.3 that the Cross Entropy needs the last output of the model to be a vector with as many dimensions as there are classes. This vector is then passed through a softmax activation to obtain the probabilities involved in the Cross Entropy. We denote this last output as $h_w(x)$. For its part, the Triplet Loss needs embeddings as inputs. The vector $h_w(x)$ is not adapted to serve as an embedding. We thus use the output from an earlier layer as input for the Triplet Loss and denote it e^{ML}.

To build the head of the network we thus need at least a pooling operation, a layer of which the outputs are the embeddings e^{ML}, and a layer of outputs for classification $h_w(x)$. We also add an intermediate layer between these two. We thus obtain exactly the same head architecture as the one of the CL model. Even though its head uses exactly the same layers, they have different purposes due to the new training objective. Figure 20 represents the new head and and the output of each layer.

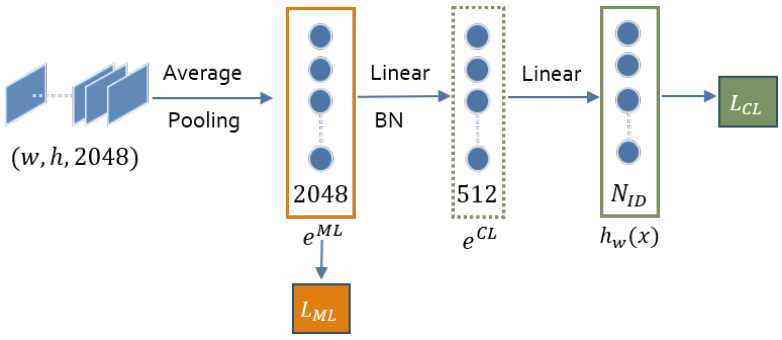


Figure 20: The Head of the CL+ML model

5.7.2 Two sets of embeddings

It was mentioned in section 5.3 that adding the intermediate layer yielded better results for the classification model. The outputs of this intermediate layer are used as embeddings for the classification model. This model uses the outputs of this layer the same way, and the embeddings it outputs will be denoted e^{CL}. For testing experiments, the model thus outputs two sets of embeddings: e^{CL} which are more strongly influenced by classification, and e^{ML} which are more strongly influenced by metric learning.

All the developments for classification and metric learning introduced in previous sections are used on their respective objective. This means that the last stride parameter of the ResNet-50 is reduced to 1, that label smoothing is applied to the Cross-Entropy loss, that the Triplet Loss uses the soft-margin version of the Hinge function and that its input triplets are batch hard triplets.

5.8 Global overview of the models

This chapter introduced the models implemented for the purposes of this thesis. It started by presenting two models that meet the Re-ID objectives (Traditional and Pair Re-ID) of learning a feature space where distances between embeddings are distributed in a given way. The CL model learns that feature space through a training supervised with a classification objective, while the ML model learns it directly with a training supervised with a metric learning objective. Both these models are CNNs that follow a two-part architecture. The backbone of the models is a ResNet-50 and the heads are series of fully connected layers that are adapted to each model.

We have seen that both models could be efficient for Pair Re-ID. The CL model has proven to be effective for Traditional Re-ID while the ML model, which is less effective for this latter task, is trained to meet the Pair Re-ID objective directly. These two models differ mostly by their supervision. We thus gave special attention to the design of their loss functions: the Cross Entropy and the Triplet Loss.

From those models two other models that combine both supervisions have been developed. The CL2ML model uses the classification and metric learning supervisions successively to train two inner models. The CL+ML model uses the classification and metric learning supervisions simultaneously and outputs two sets of embeddings, one more strongly influenced by classification and the other one by metric learning.

Alongside the discussion on the models and the supervisions, we also introduced some techniques that can make the models more efficient. We introduced the concept of label smoothing on the Cross Entropy. We also discussed an online mining technique for the Triplet Loss that feeds it batch hard triplets.

The relationships between the different models and the techniques associated with each one of those models are represented in figure 21:

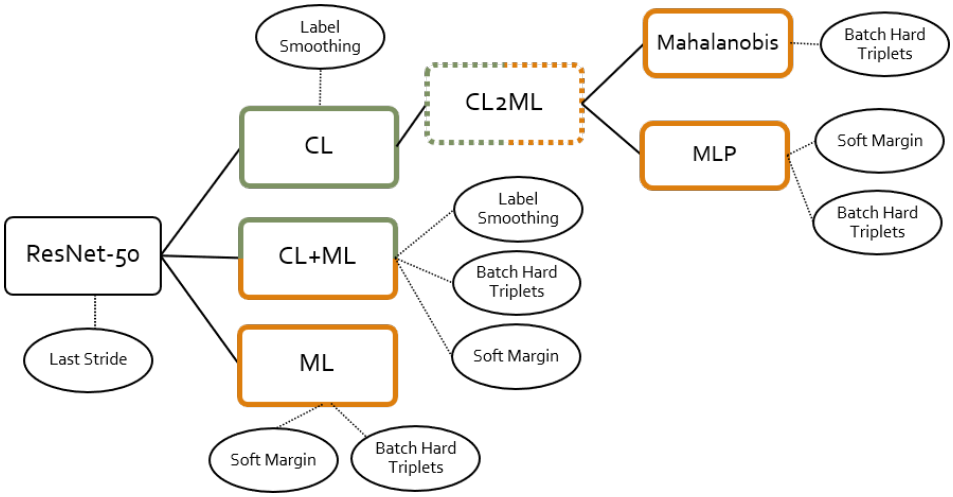


Figure 21: Relationship between the models and associated training techniques

Chapter 6 will describe the experiments that have been carried out to train and test those models for Re-ID. Chapter 7 will present the results of these experiments and draw conclusion on which model is the most efficient for Pair Re-ID, on how that relates to Traditional Re-ID and on the influence of the supervision on the performances.

Chapter 6

Experiments

Introduction

Chapter 5 described the models that have been developed for Pair Re-ID. The goal of the last two chapters is to compare and contrast the performances of those models for Pair Re-ID. In order to do so training and testing experiments need to be devised. This chapter describes how these experiments have been carried out for the purposes of this thesis. It starts by discussing the dataset Market-1501 that provides the data for all the experiments. Then it describes the training procedures and some model independent techniques that have been applied to ensure that the gradient descent converges. Techniques associated with a given model (or models) such as the mining of batch hard triplets have been described in chapter 5. Finally the testing experiments and the metrics for Pair Re-ID and Traditional Re-ID will be analysed.

6.1 Market-1501

The experiments have all been carried out on the dataset Market-1501. It is a publicly available dataset constructed specifically for Person Re-ID by *Liang Zheng et-al* [44]. This section will not go into details on the construction of the dataset as it was not part of the work of the thesis but will give some practical information about how it can be used and what it contains. The reader should keep in mind the discussion of chapter 3 on the creation of a dataset from raw data. It will be referenced punctually.

6.1.1 Bounding box images of persons identities

The dataset contains 32 668 images of 1501 identities, hence its name. Those images have been taken outside a supermarket on the campus of Tsinghua University (China). They were taken from a total of six cameras including five high resolution cameras and one low resolution. Each identity is at least present in images from two cameras to ensure that cross-camera search is possible. This is an example of ensuring the consistency and density of the dataset and part of the data evaluation procedure described in section 3.2.

Every image is doubly labelled: once regarding its identity and once regarding the camera it was taken with. These labels are referenced in the filename of the image as on the example in figure 22 where **0272** is the identity and **c3** is the camera. The rest of the filename references other information that are not useful for our purposes.

An image of an identity is actually an image of a bounding box around the identity. The bounding boxes are obtained with a detector called the *Deformable-Part-Model*

Figure 22: Example of an image of the Market 1501 Dataset [44]:
0272_c3s1_065142_02.jpg

(DPM). They are then manually annotated with the corresponding identity. Not all bounding boxes obtained with the DPM detector are associated with an identity. Indeed, the DPM detector can sometimes detect only parts of a person. Some other times, the box it produces can also contain another person or an object. To decide how to annotate a bounding box (bbox) the creators of Market-1501 came up with the following rule (*Liang Zheng et-al*, [44]):

> ... for each detected bbox to be annotated, a hand-drawn ground truth bbox is provided. For the detected and hand-drawn bboxes, the ratio of the overlapping area to the union area is calculated. In our dataset, if the area ratio is larger than 50%,the DPM bbox is marked as "good" (a routine in object detection); if the ratio is smaller than 20%, the DPM bbox is marked as "distractor"; otherwise, the bbox is marked as "junk", meaning that this image is of zero influence to re-id accuracy

The "good" images are thus annotated with their corresponding identity. Sometimes a good box contains the person only as on figure 22. Some other times, even those "good" boxes can contain objects and/or other identities as well. Figure 23 shows a good example of such a case.

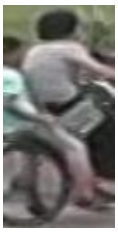


Figure 23: A "good" bounding box with other objects ([44])

Those other objects or identities can have a big impact on how the model learns, as was discussed in section 3.3. Indeed, if the model is presented with an image such as figure 23, there is no guarantee that it will learn to represent the woman (the right identity) rather than the whole of the woman, the child, and the bike.

6.1.2 Data Splits

To evaluate the performance of a model, it is recommended to split the available data to keep a portion of it for the performance evaluation. Section 3.4 explained that the standard procedure is to form three sets: a training set, a validation set and a testing set. The images of Market-1501 come already split in three sets: "Bbox train" containing 16 384 images of 750 identities, "query" containing 3368 images of 750 identities with a maximum of 6 images per identity, and "Bbox test" containing 19 732 images of the same 750 identities as the query set, of distractor boxes, and of junk boxes.

These do not correspond to the three standard sets aforementioned. "Bbox train" corresponds to the training and validation sets together and "query" and "bbox test" together form the testing set. This terminology draws from the Traditional Re-ID task: *Find, in a gallery of images, the n closest images to a query image.* The testing set is thus here separated between gallery images (bbox test) and query images. There is no validation set yet. We decided to keep about a fifth of the training data as validation. Naturally the validation set should contain examples of the 751 identities of the training set. To ensure this, every fifth picture of a given identity was placed in the validation set (starting with the first one). After splitting "bbox train" into training and validation and renaming other sets, the dataset is separated in the following sub-sets:

- A training set containing 12 185 images of 751 identities.
- A validation set containing 4199 images of the same 751 identities.
- A gallery set containing 19 732 images. These are images of 750 other identities, of the junk bounding boxes, and of the distractor bounding boxes.
- A query set containing 3368 images of the same 750 identities.

6.2 Training procedures

Chapter 4 described how neural networks are trained with gradient descent based optimisers that decrease the value of the loss function. Section 4.4 outlined that the minimisation of the loss functions has no convergence guarantee and that many problem can occur during the gradient descent. To ensure that the descent converges towards a "good" set of weights, one needs to devise training procedures. This section describes the procedures used to train the models presented in chapter 5. We use many of the solutions introduced in section 4.4. We will not focus in this chapter on why they work, but on how they have been implemented. For a theoretical overview of the techniques, the reader is referred to section 4.4.

6.2.1 Transfer learning

Section 4.4 explained that using transfer learning makes it possible to reach state of the art performances with limited time and computing resources. We recall that the idea of transfer learning is to reuse the weights of a model, trained on another dataset and/or for another task, as a starting point for the gradient descent on our own model. The Market-1501 dataset does not posses enough data to train models with as many parameters as ours from scratch. Using transfer learning is thus unavoidable.

All our models for Pair Re-ID use a ResNet-50 as a backbone (see chapter 5) and we designed a new head for each model. The ResNet-50 model trained for recognition on the dataset Imagenet (+10M images) [14] is publicly available online. We thus used it as the starting point for all our training procedures. As we design new heads, the training starts by a warmup phase of 10 epochs when only the new head is trained, so that the weights of the backbone transferred from ImageNet are not destroyed (see section 4.4). We then fine-tune the whole networks for 250 epochs.

6.2.2 Learning rate decay

To reach (locally) optimal values of the weights the gradient descent must often use decaying learning rates. Indeed, once the descent has moved from the initial values and reached a good region of the space of weights, a smaller learning rate is required to continue to descent towards a locally optimal set, and not oscillate between sub-optimals values.

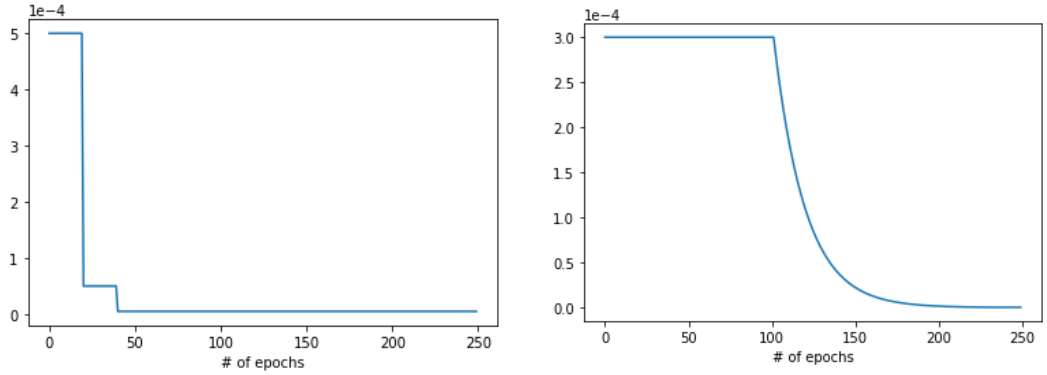
Learning rate decay is incorporated in the training of every Convolutional Neural Network implemented for the purposes of this thesis. Two decaying strategies are devised:

- The first one is used when a CNN is trained with classification supervision. The initial learning rate is $5\ 10^{-4}$. It follows then a step decay by 10 at epochs 20 and 40 of the 250 epoch training. This strategy is inspired by *George Adaimi et-al* [36]. The resulting decaying learning rate is represented in figure 24a.

- The second one is devised for training CNNs supervised with metric learning supervision. It is an exponential decay starting at epoch t_0 and ending at epoch t_1 from an initial learning rate e_0 equal to $3\ 10^{-4}$. The learning rate evolves as:

$$e(t) = \begin{cases} e_0 & \text{if } t \leq t_0 \\ e_0\ (0.001)^{\frac{t-t_0}{t_1-t_0}} & \text{if } t_0 \leq t \leq t_1 \end{cases} \tag{27}$$

  with $t_0 = 130$ and $t_1 = 200$. This strategy is inspired by *Alexander Hermans et-al* [41]. The resulting decaying learning rate is represented in figure 24b.



(a) For training supervised by classification (b) For training supervised by metric learning

Figure 24: The Learning Rate decay strategies

6.2.3 Data augmentation

Data augmentation enables to combat overfitting by adding artificial noise to our training data (as explained in section 4.4). There exist many augmentation techniques. They can be combined in numerous ways to build a data augmentation procedure. For the purposes of this thesis the following techniques have been combined:

- The images are resized to $(256, 128)$ (height, width) then pad by 10 on all borders with zeros. The resulting image is randomly cropped back to $(256, 128)$.
- The images are flipped horizontally with a probability of 50 %.
- A rectangle with a random size between 2% and 20% of the picture is erased from the original picture with a probability of 50%. Each erased pixel is replaced by a random RGB value.
- The RGB channels are rescaled to $[0, 1]$ then normalised with means $(0.485, 0.456, 0.406)$ and standard deviations $(0.229, 0.224, 0.225)$ for the red, green and blue channels respectively. These means and standard deviations are provided for models pre-trained on ImageNet.

This augmentation procedure is inspired by *George Adaimi et-al* [36]. Example pictures that show the effects of the data augmentation are given on figure 25.

6.2.4 Other training parameters

We will just list the other parameters needed to implement the exact training procedure:

Figure 25: Effects of the data augmentation procedure

- The optimiser is Adam (developed by *D.P. Kingma et-al* [45]). It uses the momentum update introduced in section 4.4.
- The batch size for the trainings supervised by classification is 32. For the metric learning supervised trainings, we explained in section 5.4 that the batches were formed of K pictures of P identities, and thus contain $P \times K$ images. In our case P is 18 and K is 4.

6.3 Testing procedures

This section describes the experiments that have been carried out to validate that the model is learning and to test how the learned knowledge can help solve Re-ID tasks. To validate that a model is truly learning during training and not overfitting on the training data the evolution of the loss function on the validation data was constantly monitored. In the case of trainings supervised by classification the accuracy was also monitored.

Even if observing that the value of the loss function decreases on the validation set is a good indicator that the model is learning, it does not provide any measurement of its ability for Re-ID tasks. In order to do so specific metrics need to be devised. Those metrics are then averaged on the testing set. We recall that this set contains images of 750 identities that are different from the 751 identities of the training and validation sets. The metrics devised for Traditional Re-ID and Pair Re-ID are not the same as the objectives also differ.

6.3.1 Traditional Re-ID metrics

The goal of Traditional Re-ID is: *Find, in a gallery of images, the n closest images to a query image*. The metrics are thus designed according to that goal. Two metrics are widely used for Traditional Re-ID: The **Cumulative Matching Characteristics (CMC) curve** and the **mean Average Precision (mAP)**.

The CMC curve shows, for increasing values of k, the probability that the identity of a query appears in the k closest images from the gallery. It is thus a step function.

Rather than the curve itself, its value for different k (typically 1 and 5) is often employed as metric. The computation for each k is very simple. For a given query image:

$$\text{cmc}_k = \begin{cases} 1 & \text{if top-k ranked gallery samples contain the query identity} \\ 0 & \text{otherwise} \end{cases} \tag{28}$$

The CMC curve for each k is obtained by averaging over all queries. The cmc_k measures if the model finds an image of the same ID as the query in the k closest images but does not care for the place of that image in those k first images. Furthermore, if there are multiple gallery images of the same identity as the query, only the position of the first one matters.

The mAP on the other hand takes those two criterion into account. Indeed, for a given query image the average precision is:

$$\begin{aligned} \text{AP} &= \frac{1}{N_{\text{TP}}} \sum_{k=1}^{N_G} \frac{\text{TP}_k}{k} \\ \text{and } \text{TP}_k &= \begin{cases} \#\text{ images of query ID in the } k \text{ first images} & \text{if image } k \in \text{query ID} \\ 0 & \text{otherwise} \end{cases} \end{aligned} \tag{29}$$

where N_G is the number of images in the gallery, and N_{TP} is the number of images of the query ID in the gallery. The mAP is the average of the AP over all queries.

To compute the CMC and mAP the gallery images are sorted according to their distances with each query image. Gallery images from the same identity and same camera as the query image are not taken into account because they are deemed too easy.

6.3.2 Pair Re-ID metrics

The objective of Pair Re-ID is that the model *learns a feature space where pairs of embeddings of the same identity are separated by a smaller distance than pairs of embeddings of different identities*. To measure how effective a model is in this task, this thesis used a measure of the separation between the distribution of same ID distances and the distribution of different ID distances. The separation between the two distributions is measured by a criterion called the **Strictly Standardised Mean Difference (SSMD)**:

$$\text{SSMD} = \frac{\mu_{\text{same ID}} - \mu_{\text{different ID}}}{\sqrt{\sigma^2_{\text{same ID}} + \sigma^2_{\text{different ID}}}} \tag{30}$$

where μ is the mean of a distribution and σ is the standard deviation. The bigger this separation between distributions, the better the performance of the model for Pair Re-ID.

Not all possible pairs of images from the testing set have been considered. Indeed, only pairs containing a query image and a gallery image are considered. There is no

theoretical justification for this but this choice already yields 49 547 pairs of images of the same identity and 53 405 700 of pairs of different identities, which was considered enough for a fair assessment. Furthermore, as was the case for the Traditional Re-ID metrics, pairs of images from the same identity and taken from the same camera are deemed to easy and are thus discarded.

Summary

This chapter starts by describing the dataset employed for all trainings and testing experiments. The way the dataset uses bounding boxes as images of identities and how these boxes are labelled is described. Then the partition of the data between training, validation, and testing set is outlined. It moves on from there to describe the training procedures that have been implemented. Finally the testing procedures are detailed. The need for different metrics for Traditional and Pair Re-ID is underlined. The metrics for Traditional Re-ID: CMC curve and mAP, and the metric for Pair Re-ID: the SSMD, are described. The performances of the models introduced in section 5 on those metrics are presented in chapter 7.

Chapter 7

Results

Introduction

This chapter presents the performances of the models introduced in chapter 5 on the testing procedures described in chapter 6. Figure 26 gives an overview of the different models and of how they relate to one another. We recall that they are differentiated by to their training supervision. For the sake of clarity the two sets of embeddings e^{ML} and e^{ML} outputted by the CL+ML models will be discussed as if resulting from two CL+ML models.

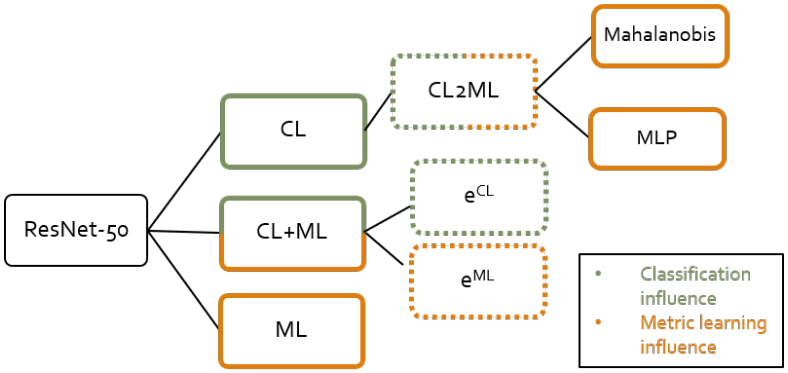


Figure 26: Overview of the models and their relations to one another

The chapter starts by presenting the performances for Traditional Re-ID. It enables to validate the quality of the models by comparing their results with state of the art models. Then it analyses the results of the testing experiments for Pair Re-ID. First by assessing how well each model did on the testing procedure in terms of SSMD. Then it checks if and how the performances for Traditional Re-ID and Pair Re-ID are correlated. The effects of the supervisions on the feature spaces learned by the models are analysed thereafter. Finally examples of pairs of images are presented for an in depth analysis. The definition of the objectives of Traditional Re-ID and Pair Re-ID can be found page 33.

7.1 Traditional Re-ID results

Section 6.3 explained that, to fit the Traditional Re-ID objective, two metrics are mainly employed: the Cumulative Matching Characteristics (CMC) curve and the mean Average Precision (mAP). We recall that the key difference between both metrics is that

mAP cares for the order in which images of the query identity are found in the sorted gallery (according to distance with the query embedding) while the CMC curve does not. The values of the CMC curve at given indexes k (CMC_k) are used as metrics rather than the curve itself. We use CMC_1 and CMC_5.

The table in figure 27 gives the performances in the aforementioned metrics for each one of our models on the testing experiment described in section 6.3. The performances of three state of the art models, one supervised with classification (in green), one with metric learning (in orange), and one which is supervised by both supervisions (similarly to the model CL+ML) are also provided.

Model	CMC@1	CMC@5	mAP
Classification (CL)	90.7%	96.7%	74.7%
Metric Learning (ML)	83.2%	93.3%	63.9%
CL2ML (MLP)	87.2%	94.9%	71.0%
CL2ML (Mahalanobis)	91.2%	96.9%	76.3%
CL+ML (e^{CL})	89.7%	96.3%	76.5%
CL+ML (e^{ML})	90.4%	96.8%	78.0%
G. Adaimi et-al [39]	91.0%	x	76.7%
A. Hermans et-al [40]	84.9%	94.2%	69.1%
H. Luo et-al [38]	94.1%	x	85.9%

Figure 27: Performances of the models on the Traditional Re-ID testing experiment

Validation of the performances The first information this table offers is that our models score in the same range than the state of the art models on all metrics. This validates the models designs and the devised training experiments. State of the art models always slightly outperform the closest corresponding model devised for the purposes of this thesis. Yet the main goal of those state of the art models is to yield the best possible performance for Traditional Re-ID. The researchers thus spent a non-negligible amount of time tuning every parameter to ensure that the model is as performant as possible. The goal of this thesis on the other hand is to compare multiple models for a new task: Pair Re-ID. The emphasis is on the comparison of the models rather than on pushing each one of them to perform to the maximum of its capacities. We have ensured that the training experiment of each model was similar. The performances presented in figure 27 validate the quality of each model.

Comparison CL and ML Our results also confirm that models in whole or in part supervised with classification outperform models supervised with metric learning for Traditional Re-ID. This observation has been commonly accepted in the Re-ID community for a couple of years [36]. This is true on all metrics: CMC_1, CMC_5 and mAP.

They also confirm that the best performances are obtained by models that combine

both supervisions. This observation has been very recently made in the Re-ID community. Until less than a year ago state of the art performances were obtained with models supervised with classification solely [36]. Our results show that all models that combine classification and metric learning, but the CL2ML - MLP model, outperform the CL model and even the state of the art classification model (*G. Adaimi et-al* [36]).

The fact that the CL2ML - MLP model is the only one of those models to yield lesser performances than the CL model is also interesting. Indeed, its architecture is such that the MLP inner metric learning model has the capacity to "overwrite" the features learned by classification quite strongly. It is probably the model the least influenced by the classification supervision among the ones that combine both supervisions.

We can also notice that the addition of the metric learning supervision improves the mAP more than it improves the CMC. From this observation we can make an assumption on how the supervision impacts the distances between query images and gallery images from the same identity. Classification brings a few images (the more resembling ones) close to the query image very well (very good CMC results with classification only). However, the more one also takes into account how close all the gallery images of the same identity are (with the mAP), the more the addition of metric learning becomes beneficial.

7.2 Pair Re-ID results

We concluded the preceding section on Traditional Re-ID by noting that *the more one also takes into account how close all the gallery images of the same identity are, the more the addition of metric learning becomes beneficial.* This consideration brings us close to the Pair Re-ID experiment where one considers the distribution of distances between all pairs of the same identity. The metric for Pair Re-ID is indeed the SSMD, introduced in section 6.3, that measures the separation between the distributions of same ID distances (same ID distribution) and different ID distances (different ID distribution). We recall that this difference should be as big as possible so that pairs of embeddings of the same ID are well separated from pairs of embeddings of different identities.

If the assumption made from the Traditional Re-ID results is confirmed, the use of metric learning, at least when combined with classification, should be effective for Pair Re-ID.

The SSMD between same ID distribution and the different ID distribution for each model is given in figure 28.

Correlation between Pair Re-ID and Traditional Re-ID The first observation to take away from those results is that models in whole or in part supervised by metric learning outperform models supervised by classification for Pair Re-ID. We recall that classification outperformed metric learning for Traditional Re-ID. We can thus not

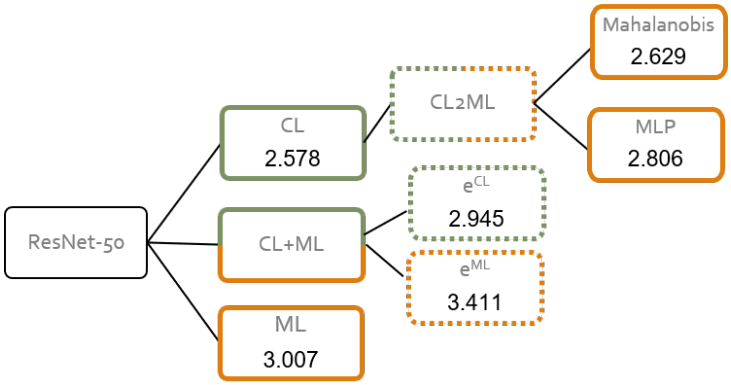


Figure 28: Stricly Standardised Mean Difference (SSMD) of the distribution of same ID distances and the distribution of different ID distances for each model

directly infer that a model will perform well for Pair Re-ID because it performs well for Traditional Re-ID. The best example is the ML model that yielded the worst performances for Traditional Re-ID and that offers the second best performances for Pair Re-ID.

We can put forward a theoretical explanation for this near reversal of roles between classification and metric learning. The metric learning supervision expressed with a Triplet Loss fits Pair Re-ID better than it does fit Traditional Re-ID. Indeed, the expression of the Triplet Loss equation 20 page 39 pushes the model towards learning same ID distances smaller by a given margin that different ID distances. This characteristic that all the same ID distances should all be smaller than the different ID distances is directly useful for Pair Re-ID and not for Traditional Re-ID.

These results are in line with the assumption, made from the Traditional Re-ID results, that metric learning is useful when all the pairs of the same ID are considered rather than the few closest pairs to the query.

Blending ML and CL Blending classification and metric learning offers mixed results compared with using metric learning only. All models that blend classification and metric learning, but CL+ML - e^{ML}, are outperformed by the ML model. The first assumption that could be made is that using classification degrades the metric learning performances. However this is probably incorrect.

Indeed, for all the models that use both supervisions (but CL+ML with the e^{ML} embeddings), the metric learning supervision comes "on top" of the classification. The reason for this is that the initial goal of this thesis was to build a model for Pair Re-ID on top of the best model for traditional Re-ID, i.e a model with classification supervision. Many models have thus a strong classification baseline and the addition of metric learning helps improve their performances for Pair Re-ID but not enough to catch up with a model that is more strongly influenced by metric learning.

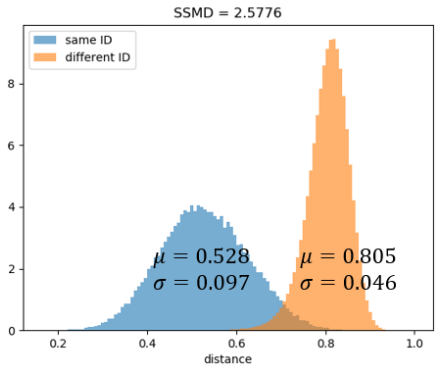


Figure 29: Distributions: CL model

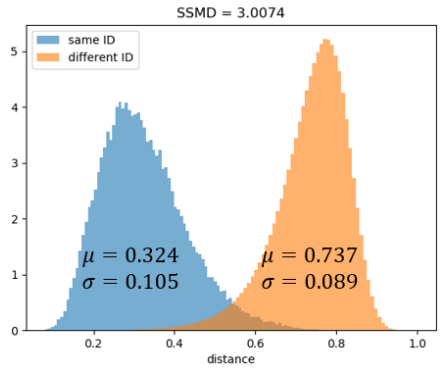


Figure 30: Distributions: ML model

Moreover, the model CL+ML - e^{ML} yields the best performances of all models. It uses the same metric learning baseline as the ML model but is also influenced by classification. In that case using classification betters the Pair Re-ID performances.

7.3 Impact of the supervision on the distributions

We analysed the performances of the different models for Pair Re-ID through the SSMD in the preceding section. It is recalled that this metric computes the separation between two distributions of distances: the distances between pairs of embeddings of the same identity and the distances between pairs of embeddings of different identities. Those distributions should be as separated as possible for Pair Re-ID.

This section focuses on the distributions, compares their characteristics for each model and will try to extract some pattern relating to the supervision by classification or metric learning.

The firsts distributions we will analyse are the one produced by the CL and ML models (noted CL and ML distributions respectively). Those two models are the baselines for classification and metric learning respectively, in that they are trained with a single type of supervision. They are thus a good starting point for the analysis. Figures 29 and 30 show the distributions for the ML and CL model respectively.

From the distributions produced by those two models we extract two observations:

- The mean of the same ID distribution (μ_{sameID}) from the ML model is considerably smaller than the one from the CL model.
- The standard deviation of the different ID distribution (σ_{diffID}) from the CL model is considerably smaller than the one from the ML model and than the same ID distribution standard deviation (σ_{sameID}) of both models.

The variation of the other parameters (μ_{diffID}, σ_{sameID}) is too small to be associated for sure to the models.

The distributions outputted by the models that combine classification and metric learning will now be analysed to try and confirm those observations and see if one can extract other impacts of the supervisions on the distributions.

The models CL2ML - Mahalanobis, and CL+ML - e^{CL} are *a priori* more strongly influenced by classification whereas the model CL+ML - e^{ML} is *a priori* more strongly influenced by metric learning. From its architecture the model CL2ML - MLP should be more strongly influenced by classification but we noticed in section 7.1 that the MLP supervised with metric learning probably overwrites the classification features quite strongly. It will thus be interesting to observe the shape of the distributions the CL2ML - MLP model produces. Figures 31 to 34 show the distributions of the models that combine both supervisions.

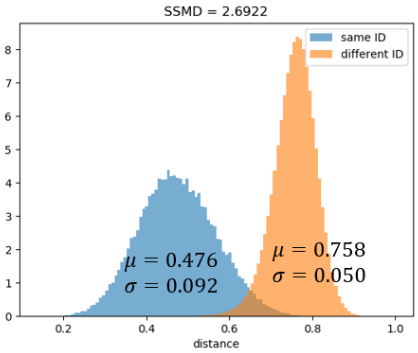


Figure 31: CL2ML - Mahalanobis

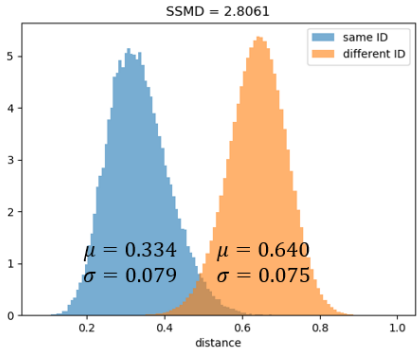


Figure 32: CL2ML - MLP

The first observation is that the assumption that the CL2ML - MLP model is strongly influenced by metric learning is confirmed. Indeed, the model outputs distributions

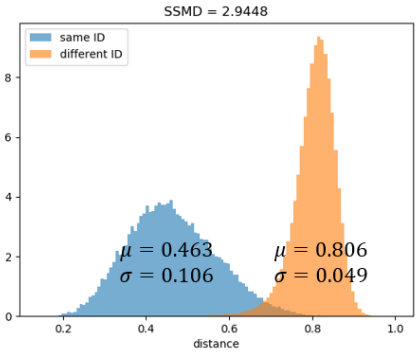


Figure 33: CL+ML - e^{CL}

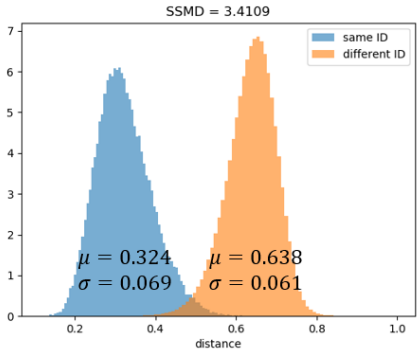


Figure 34: CL+ML - e^{ML}

similar to the other models more strongly influenced by metric learning. We will associate it with those models from now on.

Then one can observe that the assumptions made from the distributions of the CL and ML models are confirmed by the distributions of these four models. These new results also allow us to make assumptions on the two other parameters for which the variation between the CL and ML models were inconclusive:

- The mean of the different ID distributions (μ_{difID}) from the models more strongly influenced by classification is higher than the one from the models more strongly influenced by metric learning.
- The standard deviation of the same ID distributions (σ_{sameID}) from the models more strongly influenced by metric learning is lower than the one from the models more strongly influenced by classification. This assumption is surprisingly not verified by the ML model.

The means and standard deviations of the two distributions for each model are summarised in figure 35. The supervision that affects a model the most is reminded by the usual colour code. We can conclude by noting that, ideally for Pair Re-ID, one would train a model that learns the same ID distributions outputted by models more strongly influenced by metric learning and the different ID distributions outputted by the models more strongly influenced by classification.

Model	μ_{sameID}	σ_{sameID}	μ_{diffID}	σ_{diffID}
Classification (CL)	0.528	0.097	0.805	0.046
Metric Learning (ML)	0.324	0.105	0.737	0.089
CL2ML (MLP)	0.334	0.079	0.640	0.075
CL2ML (Mahalanobis)	0.476	0.092	0.758	0.050
CL+ML (e^{CL})	0.463	0.106	0.806	0.049
CL+ML (e^{ML})	0.324	0.069	0.638	0.061

Figure 35: Means and standard deviations of the same ID distances distribution and the different ID distribution for each model

7.4 Difficult pairs

This section goes back to the Pair Re-ID question: *Is this pair of images from the same person identity ?* Previous sections analysed how well the models could answer it through the distributions of distances between pairs of embeddings. Figures 29 to 34 show that no models were able to completely separate the same ID distribution from the different ID distribution. There are thus pairs of both types for which it is difficult to answer the Pair Re-ID question. We will denote them difficult pairs.

This section will give examples of difficult pairs and discuss why there are difficult. For that we consider the pairs corresponding to the left-most 1% distances of the different ID distribution and to the right-most 1% distances of the same ID distribution (the distributions computed by the CL+ML-e^{CL} are used). Figure 36 shows 16 of the pairs from the same ID distribution chosen at random. We remind the reader that those are pairs of images of the same identity for which the model has computed a big distance. It is thus difficult for it to say that it is the same person in both images. The distance between the images and an arbitrary numbering are written on top of each picture.

Figure 36: 16 pairs of images from the same identity with a big distance between the images

There are three main problematic situations that emerge from the images of figure 36. The first one is when one of the images of the pair shows the person on a bike or when his legs are obstructed by a bike. This corresponds to pairs 0, 1, 2, 3, 4, 5, 6, 9, 11. The second one is when part of the person is obstructed by an accessory of some sort that modifies the colour or the shape of the person in one of the pictures. This situation occurs in pairs 2, 5, 7, 12, 14. The last one is when the quality or luminosity of the two pictures is radically different such as in pairs 9, 11, 13, 15. Some of the pictures combine all situations. Pair 8 obviously consists in images of two different identities. This is because some of the pictures have been wrongly labelled by the constructor of the dataset. I considered removing them manually from the set but there are too many of them.

Figure 37 shows 16 difficult pairs (chosen at random) from the different ID distribution. We remind the reader that those are pairs of images of different identities for which the model has computed a relatively small distance. It is thus difficult for the model to say that it is not the same person in both images.

Figure 37: 16 pairs of images of different identities with a small distance between the images

We can extract from figure 37 two reasons why pairs of different identities might be difficult. First the people can have similar outstanding characteristics. For example pair 6 shows two people with white shirts and a bag, and pair 15 shows two ladies in pink dresses. Pairs 0, 1, 3, 4, 5, 8, 11, 12, 14 are also difficult for that reason. Even if those persons do not look that much alike, the common defining characteristic makes the pair hard for the model. On the other hand, the persons in pairs 2, 7, 9 and 10 are truly hard to differentiate, even for a human. It is thus no surprise that the models struggle. There is no clear explanation regarding pair 13.

Figures 36 and 37 show that the pairs for which it is difficult for the model to answer Re-ID questions are difficult for an identifiable reason. This reason is clear by looking at the pair and could in some cases also present a challenge for humans. More examples of difficult pairs can be found in appendix B.

Summary

This chapter analysed the performances of the models on the testing experiments described in section 6.3. Section 7.1 presented the results for Traditional Re-ID. Those validated the designs and training experiments of the models. They also confirmed that classification supervision is more appropriate than metric learning supervision for Traditional Re-ID, but that state of the art performances can be obtained by combining both supervisions. This is particularly true when one considers how close all the gallery images of the same identity as the query are from the query.

From there the performances for Pair Re-ID were analysed according to the Strictly Standardised Mean Difference. We concluded from the results that the metric learning supervision is more efficient than the classification supervision for Pair Re-ID, that using it in on top of a strong classification baseline improves the performances compared with using classification only, and that using classification on top of a strong metric learning baseline yields the best performances.

The chapter also looked at the characteristics of the distributions of same ID distances and different ID distances. Those help us understand the results for the SSMD and showed that metric learning supervision and classification supervision compute distributions with very different shapes. Finally examples of difficult pairs were given and helped to notice that there are clearly identifiable reasons why the models struggle for those pairs.

Conclusion

8.1 Summary of the work

Throughout this work we gradually constructed a comparative analysis of deep learning models for Pair Re-ID. The use of deep learning for recognition problems such as Re-Identification is no novelty. Chapter 2 indeed explained that deep learning models are now the preferred method for solving recognition problems, predominantly thanks to their ability to learn a complex feature space to represent images. They outperform other techniques by a large margin and perform in the same range of performances as humans.

The models developed for the purposes of the thesis [1] are in line with state of the art models for (Traditional) Re-Identification. They are Convolutional Neural Networks that use a standard set of tools: convolution layers, pooling layers, dropout, batch normalisation, etc to transform pictures into embeddings (feature vectors). Those tools are described in chapter 4. The models learn from the *Market-1501* dataset [44] which was specifically design for Re-Identification. The design of the dataset has an influence on the learning of the model notably regarding the use of bounding boxes for representing images of persons. The questions and challenges regarding the construction of a dataset are outlined in chapter 3 and the learning experiments based on *Market-1501* are described in chapter 6. Those experiments use a variety of proven techniques to ensure the convergence of the training of the models: transfer learning, learning rate decay, data augmentation, etc.

The novelty of this thesis firstly comes from the perspective from which the analysis was performed. The Pair Re-ID question (see definition page 1) had not yet been discussed in the literature. Evaluating models in regard to their performances in answering that question changes the objective the models should fulfil. To be effective for Pair Re-ID a model should *learn a feature space where pairs of embeddings of the same identity are separated by a smaller distance than pairs of embeddings of different identities.* Whereas for Traditional Re-ID (see definition page 2) they should *learn a feature space where the n closest embeddings to a query image of a given identity are from the same identity.*

The Pair Re-ID objective means that all two images of a given identity should be closer in the feature space than pairs of images of different identities. To measure how the feature space learned by a model fulfils this objective we introduced a metric that has not yet been used for Re-Identification: the *Strictly Standardised Mean difference.* It measures the separation between the distribution of distances between pairs of the same identity and the distribution of distances between pairs of different identities. Its mathematical expression is given in equation 30 page 53. The shapes of the distributions

[1] the code to reproduce the developments presented in the thesis can be found at `https://github.com/bastiennNB/Pair_ReID`

(means and standard deviations) were also analysed for each model.

8.2 Contributions

As stated above, the models implemented for the purposes of this thesis are Convolutional Neural Networks. They all have a similar architecture and differ in their training supervision. We used four types of supervision: classification, metric learning, a combination of both in succession (classification then metric learning) and a combination of both simultaneously. All our models score in the same range of performances as state of the art models for Traditional Re-ID.

The analysis of the performances of the models, on testing experiments for Traditional Re-ID and Pair Re-ID, enabled us to draw conclusions regarding the effects and efficiency of the supervisions on the feature space learned by the models, and on the shapes of the distributions of distances between embeddings.

Firstly the comparative analysis confirmed that classification outperforms metric learning for Traditional Re-ID, a known observation in the Re-ID community [36]. It also goes along recent works [38] [46] which stated that state of the art performances for this Traditional Re-ID task can be obtained by using a combination of classification and metric learning simultaneously. This thesis observed that combining them in succession can also improve the performances for some implementations of that successive combination. We also observed that the addition of metric learning to a model supervised by classification is mostly beneficial when one considers how close all the gallery images are from the query image. Taking this into consideration brings us closer to the framework of Pair Re-ID.

The results for Pair Re-ID in the Strictly Standardised Mean Difference (SSMD) show that, contrary to Traditional Re-ID, metric learning outperforms classification. This makes sense mathematically because the expression of the loss for metric learning (the triplet loss equation 20 page 39) directly enforces the Pair Re-ID objective (see definition page 33). Nevertheless, we observed that the best performances for Pair Re-ID were obtained by using a supervision that combines classification and metric learning simultaneously with a stronger influence of metric learning. However, we also observed that combining both supervisions with a stronger influence of classification (whether it be successively or simultaneously) is less efficient than using metric learning solely.

The analysis of the means and standard deviations of the distribution of distances between pairs of the same identity (same ID distribution) and of the distribution of distances between pairs of different identities (different ID distribution) offers more insight into the SSMD results. Globally we observed that the same ID distribution produced by models more strongly influenced by metric learning had lower means and lower standard deviations than the models more strongly influenced by classification. On the other hand, the different ID distributions produced by models more strongly influenced by classification had higher means and lower standard deviations. Ideally, one would

thus create a model that outputs the same ID distributions outputted by models more strongly influenced by metric learning and the different ID distributions outputted by the models more strongly influenced by classification.

Even the model that yielded the best performances for Pair Re-ID was not able to completely separate the same ID distribution from the different ID distribution. This means that there are some pairs for which it is still difficult for the model to determine whether they are from the same identity. A visual inspection of those pairs revealed that there are identifiable reasons which make them difficult for the model. For example a person can have a flashy accessory on in one of the images of the pair but not in the other, which changes his appearance. At the other end of the spectrum, different people can be hard to differentiate because they truly look alike even for humans.

8.3 Perspectives

The comparative analysis offered by this thesis focuses on Pair Re-ID, a question which had not yet been discussed in the literature. The analysis brought forward a broad range of results. Nevertheless, some of them reveal the need for further developments. Firstly the analysis concluded that the best supervision for Pair Re-ID was a combination of classification and metric learning simultaneously with a stronger influence of metric learning. However, only one model implements this supervision and it might be interesting to confirm this conclusion by devising a model that implements the same type of supervision with a different architecture.

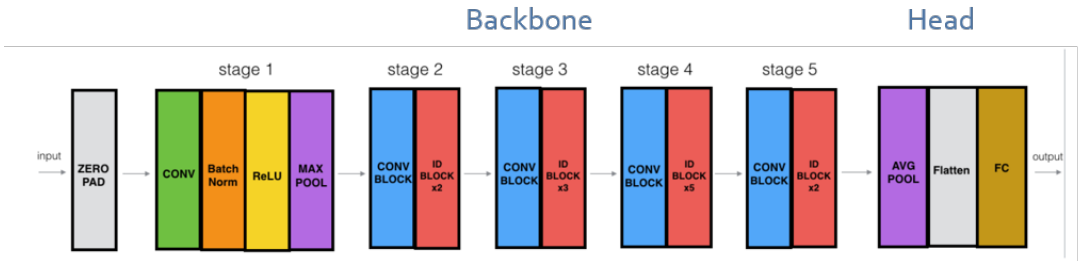
Moreover, the analysis of the shape of the distributions of distances between pairs of embeddings of same and different identities revealed that it would be ideal to devise a model that outputs same ID distributions similar to the models more strongly influenced by metric learning, and different ID distributions similar to the models more strongly influenced by classification. Designing such a model would offer great prospects for practical applications such as tracking that rely on answering the Pair Re-ID question.

Finally, one could consider a third type of supervision: similarity learning. It lies in between classification and metric learning. Indeed, a model supervised by similarity learning takes pairs of images as input and returns a binary decision on whether the pair is from the same person identity. It directly handles the Pair Re-ID question but the binary decision might not be as efficient in learning a feature space as the metric learning supervision.

This thesis proposes a new perspective on the Re-Identification field. The analysis of the models and their supervisions through Pair Re-ID brings many interesting new conclusions and provides the opportunity for further works on the topic. There is the potential for great improvement for practical applications that rely on Re-Identification.

Appendix

A ResNet-50



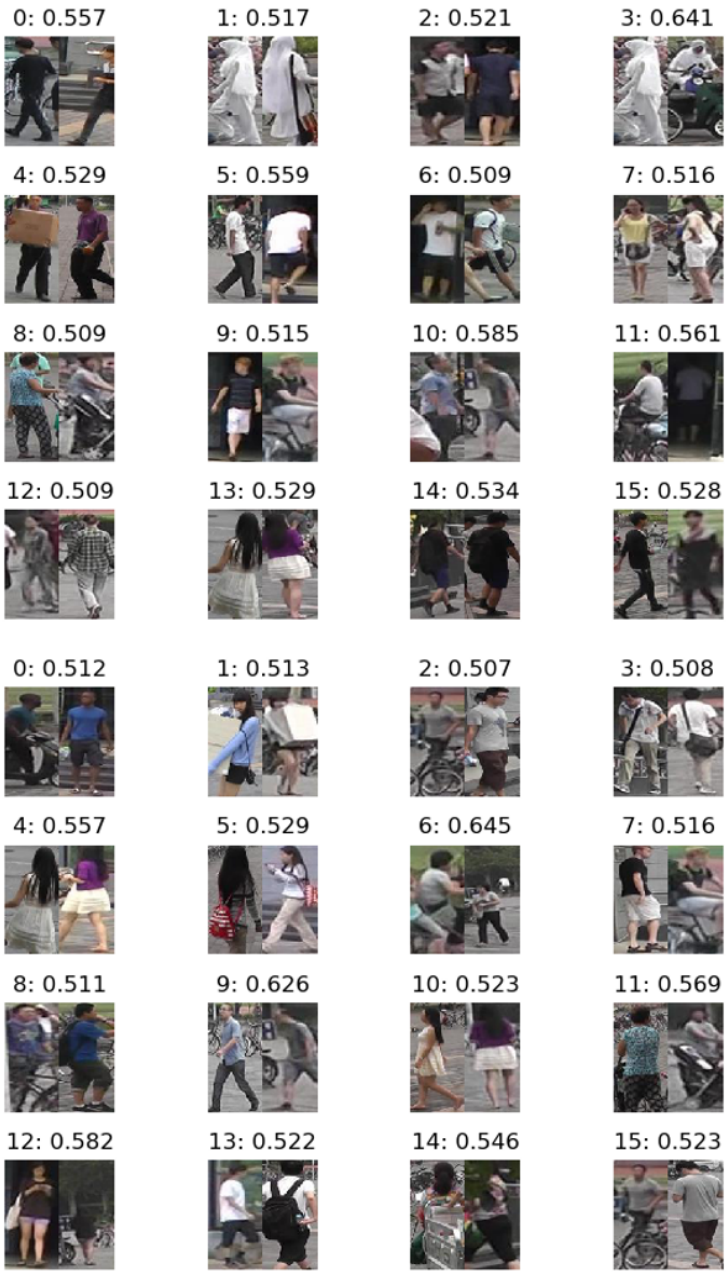
The details of this ResNet-50 model are:

- Zero-padding pads the input with a pad of (3,3)
- Stage 1:
  - The 2D Convolution has 64 filters of shape (7,7) and uses a stride of (2,2). Its name is "conv1".
  - BatchNorm is applied to the channels axis of the input.
  - MaxPooling uses a (3,3) window and a (2,2) stride.
- Stage 2:
  - The convolutional block uses three set of filters of size [64,64,256], "f" is 3, "s" is 1 and the block is "a".
  - The 2 identity blocks use three set of filters of size [64,64,256], "f" is 3 and the blocks are "b" and "c".
- Stage 3:
  - The convolutional block uses three set of filters of size [128,128,512], "f" is 3, "s" is 2 and the block is "a".
  - The 3 identity blocks use three set of filters of size [128,128,512], "f" is 3 and the blocks are "b", "c" and "d".
- Stage 4:
  - The convolutional block uses three set of filters of size [256, 256, 1024], "f" is 3, "s" is 2 and the block is "a".
  - The 5 identity blocks use three set of filters of size [256, 256, 1024], "f" is 3 and the blocks are "b", "c", "d", "e" and "f".
- Stage 5:
  - The convolutional block uses three set of filters of size [512, 512, 2048], "f" is 3, "s" is 2 and the block is "a".
  - The 2 identity blocks use three set of filters of size [512, 512, 2048], "f" is 3 and the blocks are "b" and "c".

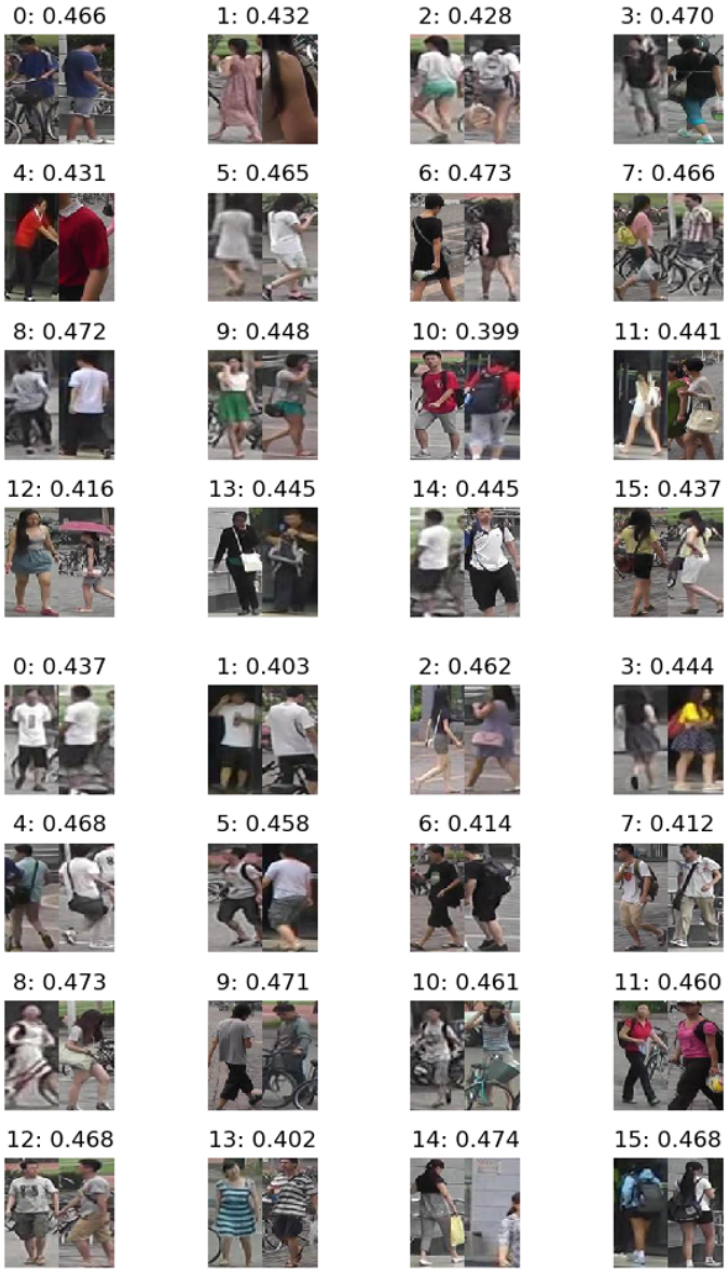
Figure 38: The ResNet-50 architecture [35]

B Difficult Pairs

B.1 Same ID pairs



B.2 Different ID pairs



References

[1] H. von Helmholtz, The Recent Progress of the Theory of Vision: A Course of Lectures Delivered in Frankfort and Heidelberg, and Republished in the Preussische Jahrbücher, 1868. Longmans, Green, 1893.

[2] J. Wilding, Perception: From Sense to Object. (Hutchinson university library), Hutchinson, 1982.

[3] K. Dawson-Howe, A Practical Introduction to Computer Vision with OpenCV. Wiley Publishing, 1st ed., 2014.

[4] J. Liter and H. Bülthoff, "An introduction to object recognition," Zeitschrift für Naturforschung. C, Journal of biosciences, vol. 53, pp. 610–21, 07 1998.

[5] I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning. MIT Press, 2016. http://www.deeplearningbook.org.

[6] D. G. Lowe, "Object recognition from local scale-invariant features," in Proceedings of the International Conference on Computer Vision-Volume 2 - Volume 2, ICCV '99, (USA), p. 1150, IEEE Computer Society, 1999.

[7] J. Josifovski, "Object recognition sift vs convolutional neural networks," 2015. Last accessed 24/04/2020
(Online) https://tams.informatik.uni-hamburg.de/lectures/2015ws/seminar/ir/pdf/slides/JosipJosifovski-Object_Recognition_SIFT_vs_Convolutional_Neural_Networks.pdf.

[8] "Descriptor matching with convolutional neural networks: a comparison to SIFT," CoRR, vol. abs/1405.5769, 2014. Withdrawn.

[9] Guru99, "Supervised vs unsupervised learning: Key differences," 2020. Last accessed 25/04/2020
(Online) https://www.guru99.com/supervised-vs-unsupervised-learning.html.

[10] A. K. Jain, R. P. W. Duin, and Jianchang Mao, "Statistical pattern recognition: a review," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 22, no. 1, pp. 4–37, 2000.

[11] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," vol. 1, pp. I–511, 02 2001.

[12] C. M. Bishop, Neural Networks for Pattern Recognition. USA: Oxford University Press, Inc., 1995.

[13] Y. Freund and R. E. Schapire, "A short introduction to boosting," in In Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, pp. 1401–1406, Morgan Kaufmann, 1999.

[14] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," International Journal of Computer Vision (IJCV), vol. 115, no. 3, pp. 211–252, 2015.

[15] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in Advances in Neural Information Processing Systems 25 (F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), pp. 1097–1105, Curran Associates, Inc., 2012.

[16] "Data preparation," in Encyclopedia of Machine Learning and Data Mining (C. Sammut and G. I. Webb, eds.), (Boston, MA), pp. 318–327, Springer US, 2017.

[17] "Algorithm noun [c]," in the Cambridge Advanced Learner's Dictionary, Cambridge University Press.

[18] H. Muller and J.-C. Freytag, "Problems, methods, and challenges in comprehensive data cleansing," 01 2003.

[19] B. Lorica, "Data collection and data markets in the age of privacy and machine learning," 2018. Last accessed 01/05/2020
(Online) https://www.oreilly.com/content/data-collection-and-data-markets-in-the-age-of-privacy-and-machine-learning/.

[20] "Ec funded caviar project/ist 2001 3754." Last accessed 02/05/2020
(Online) http://homepages.inf.ed.ac.uk/rbf/CAVIAR/.

[21] A. Ng, "Scale drives machine learning progress," in Machine Learning Yearning, deeplearning.ai, 2018.

[22] F. Pitié, "Introduction to deep learning." Trinity College Dublin Lecture, 2019.

[23] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun, "What is the best multi-stage architecture for object recognition?," in 2009 IEEE 12th International Conference on Computer Vision, ICCV 2009, pp. 2146–2153, Dec. 2009.

[24] Kurt and Hornik, "Approximation capabilities of multilayer feedforward networks," Neural Networks, vol. 4, no. 2, pp. 251–257, 1991.

[25] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," Proceedings of the IEEE, vol. 86, pp. 2278 – 2324, 12 1998.

[26] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," Nature, vol. 323, pp. 533–536, 1986.

[27] B. Polyak, "Some methods of speeding up the convergence of iteration methods," Ussr Computational Mathematics and Mathematical Physics, vol. 4, pp. 1–17, 12 1964.

[28] T. Schaul, I. Antonoglou, and D. Silver, "Unit tests for stochastic optimization," 2013.

[29] G. I. Webb, Overfitting, pp. 947–948. Boston, MA: Springer US, 2017.

[30] I. J. Good, "Explicativity: a mathematical theory of explanation with statistical applications.," Proc. R. Soc. Lond., Ser. A, vol. 354, pp. 303–330, 1977.

[31] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," CoRR, vol. abs/1207.0580, 2012.

[32] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," ArXiv, vol. abs/1502.03167, 2015.

[33] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry, "How does batch normalization help optimization?," 2018.

[34] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015.

[35] P. Dwivedi, "Residual network yourself," 2019. Last accessed 23/05/2020 (Online) https://github.com/priya-dwivedi/Deep-Learning/blob/master/resnet_keras/Residual_Networks_yourself.ipynb.

[36] G. Adaimi, S. Kreiss, and A. Alahi, "Rethinking person re-identification with confidence," 2019.

[37] Z. Zheng, L. Zheng, and Y. Yang, "A discriminatively learned cnn embedding for person reidentification," ACM Transactions on Multimedia Computing, Communications, and Applications, vol. 14, p. 1–20, Jan 2018.

[38] H. Luo, Y. Gu, X. Liao, S. Lai, and W. Jiang, "Bag of tricks and a strong baseline for deep person re-identification," in The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops, June 2019.

[39] E. Hoffer and N. Ailon, "Deep metric learning using triplet network," 2014.

[40] K. Q. Weinberger and L. K. Saul, "Distance metric learning for large margin nearest neighbor classification," J. Mach. Learn. Res., vol. 10, p. 207–244, June 2009.

[41] A. Hermans, L. Beyer, and B. Leibe, "In defense of the triplet loss for person re-identification," 2017.

[42] W. de Vazelhes, C. Carey, Y. Tang, N. Vauquier, and A. Bellet, "metric-learn: Metric Learning Algorithms in Python," tech. rep., arXiv:1908.04710, 2019.

[43] W. Li, J. Huo, Y. Shi, Y. Gao, L. Wang, and J. Luo, "Online deep metric learning," 2018.

[44] L. Zheng, L. Shen, L. Tian, S. Wang, J. Wang, and Q. Tian, "Scalable person re-identification: A benchmark," in Proceedings of the IEEE International Conference on Computer Vision, 2015.

[45] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014.

[46] M. Ye, J. Shen, G. Lin, T. Xiang, L. Shao, and S. C. H. Hoi, "Deep learning for person re-identification: A survey and outlook," 2020.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**