

HERIOT-WATT UNIVERSITY

MASTERS THESIS

---

# Multimodal Similarity Learning for Duplicate Product Identification

---

*Author:*

Thomas DI MARTINO

*Supervisor:*

Dr. Andrew IRELAND

*A thesis submitted in fulfilment of the requirements  
for the degree of MSc. in Artificial Intelligence with Speech & Multimodal  
Interaction*

*in the*

School of Mathematical and Computer Sciences

September 2020



# **Declaration of Authorship**

I, Thomas DI MARTINO, declare that this thesis titled, 'Multimodal Similarity Learning for Duplicate Product Identification' and the work presented in it is my own. I confirm that this work submitted for assessment is my own and is expressed in my own words. Any uses made within it of the works of other authors in any form (e.g., ideas, equations, figures, text, tables, programs) are properly acknowledged at any point of their use. A list of the references employed is included.

Signed:

---

Date:

---

## *Abstract*

State of the art models for Similarity Learning are all based on Deep Learning architecture using Siamese Network [Gregory et al., 2015]. They define a feature extraction pipeline that creates a latent representation of input data. This embedding vector is semantically highly descriptive and can be used for the computation of distances between data records to measure similarity. While similarity learning is a popular topic, the combination of multiple modalities has not yet been attracted most of the attention of the field.

In the context of Duplicate Product Identification, both the textual description of the product and their pictures can be used to make the similarity decision. This context of using data descriptors, e.g images and text, of different modalities require to rethink the concept of Siamese Network to perform multimodal similarity learning. In this work, multiple approaches have been explored: unimodal & multimodal Similarity Learning algorithms. The latter, combining embeddings across multiple modalities through gradient sharing method was proven to outperform any other combination of unimodal approaches through the use of N-Way & F-Beta Scoring.

Furthermore, our analysis of the impact on the learnt features of the combination of multiple modalities has given insights on how they can collaborate to optimize the training function by detecting the most informative features for each modality. Comparing the weights of a multimodal siamese network to unimodal network helped to better evaluate cross-modality data profiles captured within the embeddings.

## *Acknowledgements*

I wish to show my gratitude to Dr Andrew IRELAND with whom I had weekly opportunities to refine and judge my new advances in the writing of this Research Report. Every suggestion helped me to make this report better before concluding with this final version. Additionally, I would like to thank Dr Oliver LEMON for his highly useful feedback on my initial research report that helped me to guide my research for the last few months.

I also would like to thank the DataLab team that did an astounding work at finding me this ideal placement, especially Ms Bethany RODGERS-RINTOUL who was dedicated to finding for me and my classmates industrial placements.

Besides, I want to thank my friends from the MSc in Artificial Intelligence promotion, Dimitri, Clarence, Antoine & Alice for the highly stimulating scientific exchanges with regards to artificial intelligence, deep learning and information processing in general.

Moreover, I show a lot of appreciation to Diana Tumashkina for her daily life during these months of work under lockdown. She has been very supportive and helpful.

Finally, I show my deepest gratitude to Jordan from E.Fundamentals and other members of the team for their help and open-mindedness during this industrial placement.

# Contents

|  |      |
|--|------|
| <b>Declaration of Authorship</b>                       | i    |
| <b>Abstract</b>  | ii   |
| <b>Acknowledgements</b>                                | iii  |
| <b>Contents</b>  | iv   |
| <b>List of Figures</b>                                 | viii |
| <b>List of Tables</b>                                  | x    |
| <b>Abbreviations</b>                                   | xi   |
| <b>1 Introduction</b>                                  | 1    |
| 1.1 Motivation . . . . .                               | 1    |
| 1.2 Objectives . . . . .                               | 2    |
| 1.3 Expected results . . . . .                         | 2    |
| 1.4 Report Structure . . . . .                         | 3    |
| <b>2 Background and Literature Review</b>              | 4    |
| 2.1 Computer Vision Introduction . . . . .             | 4    |
| 2.1.1 Concept of an image . . . . .                    | 5    |
| 2.1.1.1 Pixels, the <i>atoms</i> of an image . . . . . | 5    |
| 2.1.1.2 Images, or <i>maps</i> of pixels . . . . .     | 6    |
| 2.1.2 Challenges of image data . . . . .               | 6    |
| 2.1.2.1 Image Colour invariance . . . . .              | 6    |
| 2.1.2.2 Image Shift invariance . . . . .               | 7    |
| 2.1.3 Different types of application . . . . .         | 7    |
| 2.2 Text Mining Introduction . . . . .                 | 9    |
| 2.2.1 Concept of <i>Text</i> . . . . .                 | 9    |
| 2.2.2 Word Vectorisation with Word2Vec . . . . .       | 10   |
| 2.2.2.1 What is Word Vectorisation ? . . . . .         | 10   |
| 2.2.2.2 Word2Vec Introduction . . . . .                | 10   |
| Word2Vec Arithmetic properties . . . . .               | 10   |
| 2.2.2.3 Word2Vec details . . . . .                     | 12   |

|   |           |
|---|-----------|
| Vocabulary  | 12        |
| Model   | 13        |
| 2.3 Machine Learning concepts   | 14        |
| 2.3.1 What is Machine Learning ?  | 14        |
| 2.3.2 Dataset   | 15        |
| 2.3.3 Data Preparation in Machine Learning  | 15        |
| 2.3.3.1 Data Collection   | 16        |
| 2.3.4 Evaluating a classification algorithm   | 16        |
| 2.4 Deep Similarity Learning  | 18        |
| 2.4.1 Deep Learning: the new generation of statistical models                               | 21        |
| 2.4.1.1 Deep Learning introduction  | 21        |
| Neural Networks, a high-end learning model  | 21        |
| 2.4.1.2 Deep Learning Models: a quick overview  | 21        |
| Dense Neural Networks   | 22        |
| Convolutional Neural Networks   | 22        |
| Convolutional Layers  | 23        |
| Pooling Layers  | 24        |
| Recurrent Neural Networks   | 25        |
| RNN Overview  | 25        |
| LSTM, a particular type of Recurrent Neural Network   | 27        |
| 2.4.1.3 Training a Deep Learning model: the Back-Propagation algorithm                      | 29        |
| 2.4.2 Similarity Learning   | 30        |
| 2.4.2.1 Design of a Deep Neural Network for Similarity Learning: the Siamese Neural Network | 31        |
| 2.4.2.2 Losses of Deep Similarity Learning  | 34        |
| Binary Cross-Entropy Loss   | 34        |
| Contrastive Loss  | 35        |
| Triplet Loss  | 36        |
| 2.4.2.3 Data Sampling: a crucial asset for Similarity Learning                              | 37        |
| Distance-weighted sampling  | 38        |
| Mini-batch-wise hard sampling   | 38        |
| 2.4.3 Related methods in Multimodal Deep Learning   | 39        |
| 2.4.3.1 Stacked-Autoencoder to learn multimodal embedding                                   | 39        |
| 2.4.3.2 Improving invariant embedding with the use of multi-task learning                   | 41        |
| 2.5 Conclusion  | 43        |
| <b>3 Problem Context, Analysis &amp; Requirements</b>                                       | <b>46</b> |
| 3.1 Problem context: E.Fundamentals & The DataLab   | 46        |
| 3.1.1 The DataLab presentation  | 46        |
| 3.1.2 E.Fundamentals presentation   | 47        |
| 3.2 Problem analysis  | 48        |
| 3.3 Project Requirements  | 48        |
| 3.3.1 Project's Goals   | 49        |
| 3.3.1.1 Stakeholders  | 49        |
| 3.3.1.2 MoSCoW Requirements   | 49        |

|  |           |
|--|-----------|
| Functional Requirements . . . . .  | 49        |
| Non-Functional Requirements . . . . .  | 50        |
| 3.3.2 Use Cases Study . . . . .  | 51        |
| 3.3.3 Evaluation . . . . .   | 53        |
| 3.3.3.1 Evaluation discussion . . . . .  | 53        |
| 3.3.3.2 Retained metrics: F-beta score & n-way testing . . . . .   | 53        |
| F-beta score . . . . .   | 53        |
| N-way testing . . . . .  | 54        |
| <b>4 Professional, Legal and Ethical Issues</b>  | <b>56</b> |
| 4.1 Professional . . . . .   | 56        |
| 4.2 Legal . . . . .  | 57        |
| 4.3 Ethical . . . . .  | 57        |
| 4.4 Social Issues . . . . .  | 57        |
| <b>5 Implementation: Dataset &amp; Models</b>  | <b>59</b> |
| 5.1 Data . . . . .   | 59        |
| 5.1.1 Data retrieval . . . . .   | 59        |
| 5.1.2 Data cleaning . . . . .  | 60        |
| Remove products with corrupted image . . . . .   | 61        |
| Remove wrongly matched product . . . . .   | 61        |
| 5.1.3 Dataset characteristics . . . . .  | 61        |
| 5.2 Models . . . . .   | 62        |
| 5.2.1 Unimodal models . . . . .  | 63        |
| 5.2.1.1 Siamese RNN . . . . .  | 63        |
| Preprocessing . . . . .  | 64        |
| Siamese LSTM Architecture . . . . .  | 64        |
| 5.2.1.2 Siamese CNN . . . . .  | 65        |
| 5.2.2 Siamese Multimodal neural network . . . . .  | 66        |
| 5.2.2.1 Architecture . . . . .   | 66        |
| 5.2.2.2 Modalities combination strategy . . . . .  | 67        |
| 5.3 Code architecture & Program overview . . . . .   | 68        |
| 5.3.1 Scripts folder details . . . . .   | 68        |
| 5.3.2 Demonstration website . . . . .  | 69        |
| <b>6 Experimental Design &amp; Results</b>   | <b>70</b> |
| 6.1 Experimental Design . . . . .  | 70        |
| 6.1.1 Software & Hardware . . . . .  | 70        |
| 6.1.2 Experimental Setup . . . . .   | 71        |
| 6.1.2.1 Data setup . . . . .   | 71        |
| 6.1.2.2 Training procedure . . . . .   | 72        |
| 6.1.3 Research Questions & Hypothesis . . . . .  | 74        |
| 6.1.3.1 Research Questions . . . . .   | 74        |
| 6.1.3.2 Hypotheses . . . . .   | 75        |
| 6.2 Results . . . . .  | 75        |
| 6.2.1 RQ1: <i>Does a deep learning model outperform simpler and more common machine learning algorithms with handcrafted features for product similarity learning? To what extent?</i> . . . . . | 75        |

|         |   |    |
|---------|---|----|
| 6.2.1.1 | Baseline Model Presentation . . . . .   | 76 |
| 6.2.1.2 | Experimental results . . . . .  | 77 |
| 6.2.2   | RQ2: <i>To what extent does having a siamese multimodal model give better performance than unimodal models? What is the impact on the resulting embedding ?</i> . . . . .               | 78 |
| 6.2.2.1 | Deep Learning model performance comparison . . . . .  | 78 |
|         | Analysis of title processing performance . . . . .  | 79 |
|         | Embedding exploration & Multimodal collaboration . . . . .  | 80 |
| 6.2.3   | RQ3: <i>How much does the Convolutional Neural Network's weights of the multimodal model differ from the weights of the same model but within the unimodal architecture ?</i> . . . . . | 83 |
| 6.2.4   | RQ4: <i>Can multitask learning improve model performance ?</i> . . . . .  | 85 |
| 6.3     | Conclusion . . . . .  | 88 |
| 7       | <b>Conclusion &amp; Future Work</b>   | 89 |
| 7.1     | Conclusion . . . . .  | 89 |
| 7.2     | Future work & reflections . . . . .   | 90 |
| A       | <b>Configuration File</b>   | 92 |
| B       | <b>Plot of Decision Trees</b>   | 94 |
|         | <b>Bibliography</b>   | 97 |

# List of Figures

|      |  |    |
|------|--|----|
| 2.1  | Graphical representation of time dimension of textual data . . . . .   | 9  |
| 2.2  | Word2Vec explicative graphic . . . . .   | 11 |
| 2.3  | Influence of vocabulary size on system accuracyChristophe et al. [2017] . .  | 12 |
| 2.4  | Graphical representation of the Skip-Gram model for the Word2Vec embedding . . . . .   | 13 |
| 2.5  | Graphical representation of a ML Pipeline in Computer Vision . . . . .   | 16 |
| 2.6  | Presentation of the confusion matrix . . . . .   | 17 |
| 2.7  | Taxonomy of Computer Vision . . . . .  | 19 |
| 2.8  | Comparison of a Deep and a Shallow Artificial Neural Network . . . . .   | 22 |
| 2.9  | Simplified convolution example . . . . .   | 23 |
| 2.10 | Average Pooling example . . . . .  | 24 |
| 2.11 | Max Pooling example . . . . .  | 25 |
| 2.12 | Simplified representation of an RNN architecture predicting the next letter in an input word splitted as a sequence of letters . . . . .     | 26 |
| 2.13 | Details of structure of an RNN network . . . . .   | 26 |
| 2.14 | Graphical representation of an LSTM cell . . . . .   | 27 |
| 2.15 | Architecture of the $G_W$ mapping function (source: [Hadsell et al., 2006]) . .  | 32 |
| 2.16 | Results of DrLIM Method on semantic clustering of MNIST dataset (source: [Hadsell et al., 2006]) . . . . .                                   | 32 |
| 2.17 | Samples from the Omniglot dataset (source: [Gregory et al., 2015]) . . . . .   | 33 |
| 2.18 | Convolutional Siamese architecture (source: [Gregory et al., 2015]) . . . . .  | 33 |
| 2.19 | Comparison of different networks against baselines on one-shot accuracy over the Omniglot dataset (source: [Gregory et al., 2015]) . . . . . | 34 |
| 2.20 | BCE Loss comparison . . . . .  | 35 |
| 2.21 | Contrastive Loss illustration . . . . .  | 35 |
| 2.22 | Triplet Loss illustration . . . . .  | 36 |
| 2.23 | Autoencoder illustration . . . . .   | 40 |
| 2.24 | The stacked-autoencoder architecture of Silberer and Lapata [2014] . . . . .   | 40 |
| 2.25 | Word-Pair similarity performance of the stacked autoencoder architecture [Silberer and Lapata, 2014] . . . . .                               | 41 |
| 2.26 | Multitask Illustration with parameter sharing (source: [Wang and Yao, 2019]) . . . . .   | 42 |
| 2.27 | Illustration of weights' role in multitask learning . . . . .  | 43 |
| 2.28 | Comparison of some negative and positive aspects of Classification tasks and Similarity Learning tasks . . . . .                             | 43 |
| 3.1  | The DataLab logo . . . . .   | 47 |
| 3.2  | The E.Fundamentals logo . . . . .  | 47 |

|      |  |    |
|------|--|----|
| 3.3  | Original Merging System Design   | 48 |
| 5.1  | Extract of the dataset   | 62 |
| 5.2  | Example of a data sample: true product on the left, extracted product on the right   | 62 |
| 5.3  | Abstract representation of the embedding extraction process  | 62 |
| 5.4  | Text Processing Pipeline   | 63 |
| 5.5  | Text Preprocessing Example   | 64 |
| 5.6  | BiLSTM architecture used   | 65 |
| 5.7  | Original VGG16 Architecture (source: [Nash et al., 2018])  | 65 |
| 5.8  | Multimodal Network architecture  | 67 |
| 5.9  | Screenshot of the demo website   | 69 |
| 6.1  | Example of data augmentation   | 72 |
| 6.2  | Online Mini-Batch hard sampling illustration   | 74 |
| 6.3  | Handcrafted features for Decision Tree Classification  | 76 |
| 6.4  | PCA Components reshaped as pixel maps  | 78 |
| 6.5  | Heatmap of the normalised LSTM output activations from the Siamese Recurrent Neural Network  | 80 |
| 6.6  | Heatmap of the normalised LSTM output activations from the Siamese Multimodal Neural Network   | 81 |
| 6.7  | Plots of t-SNE's representation of 4 embeddings  | 82 |
| 6.9  | Two numerical solutions  | 84 |
| 6.10 | Multi-Task training procedure  | 86 |
| 6.11 | Comparison of the Grad-CAM of the activation maps on a similarity task for the Siamese CNN (A) and the PreTrained Siamese CNN (B) models | 87 |
| B.1  | Decision tree exploiting the word count feature  | 94 |
| B.2  | Decision tree exploiting the image distance feature  | 95 |
| B.3  | Decision tree exploiting both features   | 96 |

# List of Tables

|     |   |    |
|-----|---|----|
| 2.1 | Description of other metrics . . . . .                                  | 18 |
| 3.1 | Functional Requirements Analysis . . . . .                              | 50 |
| 3.2 | Non-Functional Requirements Analysis . . . . .                          | 51 |
| 6.1 | Software version listing . . . . .                                      | 71 |
| 6.2 | Comparative table of GPU performance (source: lambdalabs.com) . . . . . | 71 |
| 6.3 | Performance comparison between Deep Models and Decision Trees . . . . . | 77 |
| 6.4 | Performance comparison between Deep Models . . . . .                    | 79 |
| 6.5 | Table of multimodal weights of the merging layer . . . . .              | 82 |
| 6.6 | Layer-wise weights difference . . . . .                                 | 84 |
| 6.7 | Performance comparison between Deep Models . . . . .                    | 87 |

# Abbreviations

|                       |                                     |
|-----------------------|-------------------------------------|
| <b>CNN</b>            | Convolutional Neural Network        |
| <b>DL</b>             | Deep Learning                       |
| <b>LSTM</b>           | Long Short Term Memory              |
| <b>MAE</b>            | Mean Absolute Error                 |
| <b>MedAE</b>          | Median Absolute Error               |
| <b>MSE</b>            | Mean Squared Error                  |
| <b>ML</b>             | Machine Learning                    |
| <b>RNN</b>            | Recurrent Neural Network            |
| <b>SL</b>             | Similarity Learning <b>Word2Vec</b> |
| <b>Word To Vector</b> |                                     |

# Chapter 1

## Introduction

### 1.1 Motivation

Measuring similarity between two data records is a task with a lot of applications: whether it is facial recognition [Schroff et al., 2015], object tracking in videos [Bertinetto et al., 2016] or change detection [Daudt et al., 2018], a lot of solutions already exist and are used daily (e.g. Baidu’s facial recognition system).

However, most of the existing solutions focus on a single modality and cross-modality similarity learning is not so common. It is in this context that my MSc project, as a part of an industrial placement, took place. Working within E.Fundamentals, my goal is to apply similarity learning algorithm to merge products from multiple online retailers based on how similar they are. As an e-commerce analytics company, E.Fundamentals is working with multiple data entries describing products including product images and titles. To compare the extracted products from multiple retailers, the company uses near-strict equality between products represented by their image and titles. However, E.Fundamentals misses on a lot of potential merges and uses a team of Data Engineer to merge similar products by hand when images and titles have slight differences that will not trouble human observes but can prevent classic algorithms to detect similarity (e.g. different lightning conditions, position, orientation..).

## 1.2 Objectives

In this context, the objectives of this project are to use Deep Learning algorithms, popular for their invariance to change in data, to perform similarity detection, in place of the Data Engineer team. If integrated to E.Fundamentals systems, these developed algorithms would use the two main product descriptors (image & title) to measure the similarity between different products:

- If the algorithm is confident enough, products would be merged;
- If the algorithm has doubts, human intervention will be requested;
- If the algorithm is sure about dissimilarity, products will not be merged.

To achieve the following, different sub-objectives exist:

1. Extract from E.Fundamentals database a clean and reliable dataset of merged and unmerged products;
2. Develop a simple baseline model to use as a point of comparison;
3. Develop Deep Learning architectures that could be used in the context of similarity learning;
4. Train the developed architectures with the created dataset while keeping the code and experiments compatible with E.Fundamentals workflow.

The multiple algorithms would extract information from the different modalities of the product data in both unimodal and multimodal manners. These two ways of processing data would then be benchmarked and compared.

## 1.3 Expected results

The presented task seems to have favourable conditions for Deep Learning: relatively clean data, in big quantities and a defined problem of Similarity Learning, already studied in Deep Learning. This points towards Deep Learning algorithms outperforming the baseline model. Another aspect of Deep Learning is that it is popular for efficiently

encapturing data profile spanning across multiple modalities. Hence, we expect multi-modal solutions to perform better than unimodal.

## 1.4 Report Structure

After having introduced the goal of the project and the context of the problem, we will first establish a literature survey, presenting the topic and its related relevant literature. Once the literature has been presented, we will develop a thorough requirements analysis, presenting our system architecture as well as other lower levels requirements, while distinguishing the ones that are optional from the mandatory as well as the functional from the non-functional. Then, we will evoke eventual professional, legal and ethical issues before presenting the multiple models implemented as well as the data used for experiments. Additionally, these experiments will be detailed and their context will be presented: the research questions, at the core of this project, will be explored and hypotheses will be either validated or refuted.

After presenting our main solution and discussing it, we will conclude by analysing the ins and outs of our solution as well as the unanswered questions and potential room for improvement.

# Chapter 2

## Background and Literature Review

The objective of this research project is to come up with an algorithm that will be able to compare two pieces of data, both composed of a text with an image, and to output if these objects are similar, dissimilar or if human expertise is needed. As the product is represented as a picture and a text description, the Similarity Algorithms used will need to take advantage of both of these data modalities. However, multimodal comparison requires to explore the underlying concepts of both text and image processing. For that reason, we will first introduce notions of Computer Vision, followed by key concepts of Text Mining and Natural Language Processing while, finally, talking about solutions to former will require processes of Text Mining and Natural Language Processing.

### 2.1 Computer Vision Introduction

In this section about Computer Vision concepts presentation, we will split the work in different main part:

1. Computer Vision *Definition*
2. Contextualising the blend between ML and CV (especially the recent shift in automated learning methods developed recently.)

Images are a snapshot of reality, translating into numeric figures spatial information. They are formed through the interaction of 3D scene elements, lighting and camera optics/sensors. Although artificially generated imagery does not come from the exact capturing process, it still embodies the same idea of representing spatial/physical data as a Map of pixels.

### 2.1.1 Concept of an image

#### 2.1.1.1 Pixels, the *atoms* of an image

Said pixels can have multiple channels. In a more mathematical based approach, you can note a pixel  $p$  as being:

$$p = (a_1, a_2, \dots, a_n) \text{ where } n \in \mathbb{N}$$

The number  $n$  is then called the *number of channels* of an image. The number of channels used in a computer vision application depends on multiple factors including such as:

1. how precise the application needs to be (the more channels, the more potential the application has, but it also increases complexity and power consumption);
2. the physical limits induced by the recording equipment (e.g not the same number of channels or kind of channels if using a classic video camera or an IR camera);
3. the amount of processing power at disposal.

A non-exhaustive list of different pixel channels would be the following:

1. RGB (Red, Green, Blue): using a measure of the intensity of the three primary colours for each primary colours of the light spectrum, one can recreate a large panel of colours, depending on the depth of the image (the depth is the number of bits encoding the red, blue and green values; most commonly encoded on 8 bits, meaning that we can have  $2^8$  nuances of red, of green and of blue).

2. HSV (Hue, Saturation, Value) or its variant HSL (Hue, Saturation, Lightness): In the case of the HSL, the lightness is a direct component of the pixel, while it was implicitly provided in the RGB model (higher R, G and B values would mean a higher brightness), meaning that the lightness of an image can be modified very easily and with less perturbation than in an RGB mode.

### 2.1.1.2 Images, or *maps* of pixels

An image can be seen as a 2D Map of pixels. Strictly speaking, we could define an image A of size  $n \times m$ , it can be rewritten as such:

$$A = \begin{pmatrix} p_{1,1} & \dots & p_{1,m} \\ \vdots & \ddots & \vdots \\ p_{n,1} & \dots & p_{n,m} \end{pmatrix}$$

where each  $p_{(i,j)} \forall i \in [1, n], j \in [1, m]$  represents a pixel with t channels

More than just by the values of each pixel, an image holds also a considerable amount of information through the position of each of them.

### 2.1.2 Challenges of image data

When working with image data, what may be instinctive to the human eye is not straight forward for a computer: as seen earlier, they perceive images as matrices of pixel values, which arises challenges proper to images.

#### 2.1.2.1 Image Colour invariance

Working with images is not straight forward: the same object, under different lighting conditions, will be treated differently by the same algorithm if no care is taken when either capturing or pre-processing images. Providing an invariant description of an object in an image is a challenge since many years: “invariant description” is to be understood as a universal description, that is neither dependant on the colour environment of the image (e.g variations in the colour spectrum with real red looking orange-ish) nor the

lighting conditions (e.g variations in the luminosity of the scene). In 2002 for example, [Geusebroek et al. \[2001\]](#) tried to tackle this issue by providing a physics-based framework to extract object colour properties and measurements.

### 2.1.2.2 Image Shift invariance

When classifying images, and especially when classifying objects contained in an image, it is mandatory to consider the possibility for these objects to be found in different positions in the image and at different scales. However, we would expect our classifier to perform in the same way for each of these situations. While this issue has been around for decades, it has been one of the main problems to be tackled by Convolutional Neural Networks, a particular type of Artificial Neural Network that we will discuss later on.

### 2.1.3 Different types of application

Computer Vision tasks are *very* diverse, as one may do anything and everything with an image. [Richard \[2010\]](#) explained how Computer Vision tasks can be split in the following, non-exhaustive, subcategories:

- Basic Image Processing (point operators, linear filtering, geometric transformation...)
- Feature Detection and Matching
- Segmentation
- Feature-based alignment (e.g pose estimation)
- Structure from motion (e.g triangulation)
- Dense motion (e.g estimation)
- Image stitching (e.g global alignment)
- Computational photography (e.g photometric calibration)
- Stereo correspondence (e.g. epipolar geometry)
- 3D Reconstruction

- Image-based rendering
- Recognition

## 2.2 Text Mining Introduction

Even if we could consider Text Mining data as being similar to Images in the sense that they both are examples of unstructured data, there is still a lot of differences, in what they are for a computer and their mutual underlying concepts. As the exploitation of text to compare different products can be proven useful or even essential, we will present what Text Mining is and how is text represented, computer-wise.

### 2.2.1 Concept of *Text*

In its rawest form, a text can be seen as a string, a list of individual distinct characters. A parallel could be driven with the pixels of an image that cannot be divided. However, the main difference between an image and text is their dimension: a text is uni-dimensional while an image is bi-dimensional (for 2D images). This is very important as when in an image, neighbours of pixels are situated above, below as well as on the left and right (if considering the Manhattan definition of neighbourhood, that is not the only distance available [[Sudip and Hari Sagar, 2016](#)]), in a text, the neighbours, per se, of a word are the words *before* and *after* it. The notions of “*before*” and “*after*” introduce the idea of a temporal dimension: indeed, the uni-dimensional concept of text can be seen as a temporal dimension representing the order in which the words should be read.

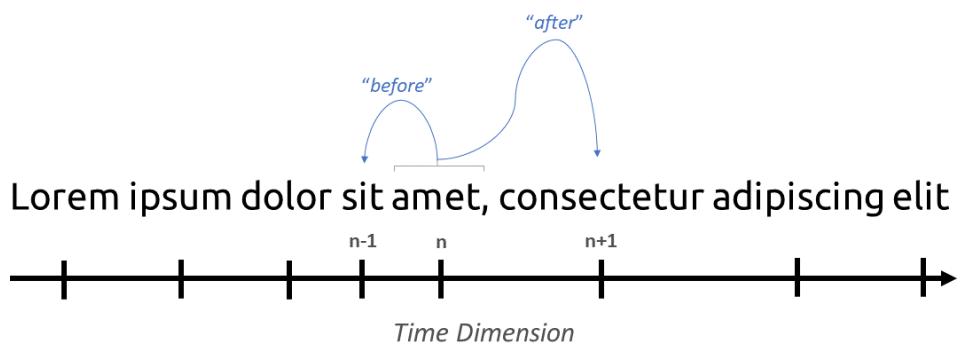


FIGURE 2.1: Graphical representation of time dimension of textual data

In this figure (cf. Figure 2.1), we see that we have split a sentence into words <sup>1</sup>. This process induces a new way to represent text: an array of words, themselves represented as strings. However, this is yet to be interpretable for a computer: a computer only works with numbers. These words will then need to be transformed into values. We will now detail a few example processes that transform a text into machine-interpretable data: vectors of numbers.

## 2.2.2 Word Vectorisation with Word2Vec

### 2.2.2.1 What is Word Vectorisation ?

The concept of Word Vectorisation is simple: given a word, we transform it into a vector of values that provides a description (e.g semantic information) of the word that can be interpreted by a computer. The final objective of such transformation is to be able to compare words between them and to have a quick similarity measure that is not only comparing how many letters they have in common but rather how close they are in meaning, etc..

We will present the most popular methods for Word Vectorisation: Word2Vec.

### 2.2.2.2 Word2Vec Introduction

Developed and presented first in [Tomas et al., 2013], the “*Word representations in Vector Space*” model helps to add a similarity measure between words (that would then eventually be extendable to compare documents that contain these words). This technique transforms a word into a vector of values where each of these value measures the word intricacies with a concept <sup>2</sup>.

**Word2Vec Arithmetic properties** The most famous example of Word2Vec results is the triplet (King, Man, Woman) where we want to have some similarities between the words “king” and “man” as they both represent the male sex while at the same time having similarities between the words “man” and “woman” as both represent, it is true,

---

<sup>1</sup>Let it be clear that the discretisation process described here is purely subjective as we, humans, consider the space as a word separator

<sup>2</sup>As this representation is statistically learnt, not all values are interpretable, but detailing it as such is more understandable

somewhat antagonist concepts but share at the same time the representative meaning of “human”. These ideas could be summed up in the following figure (cf Figure 2.2):

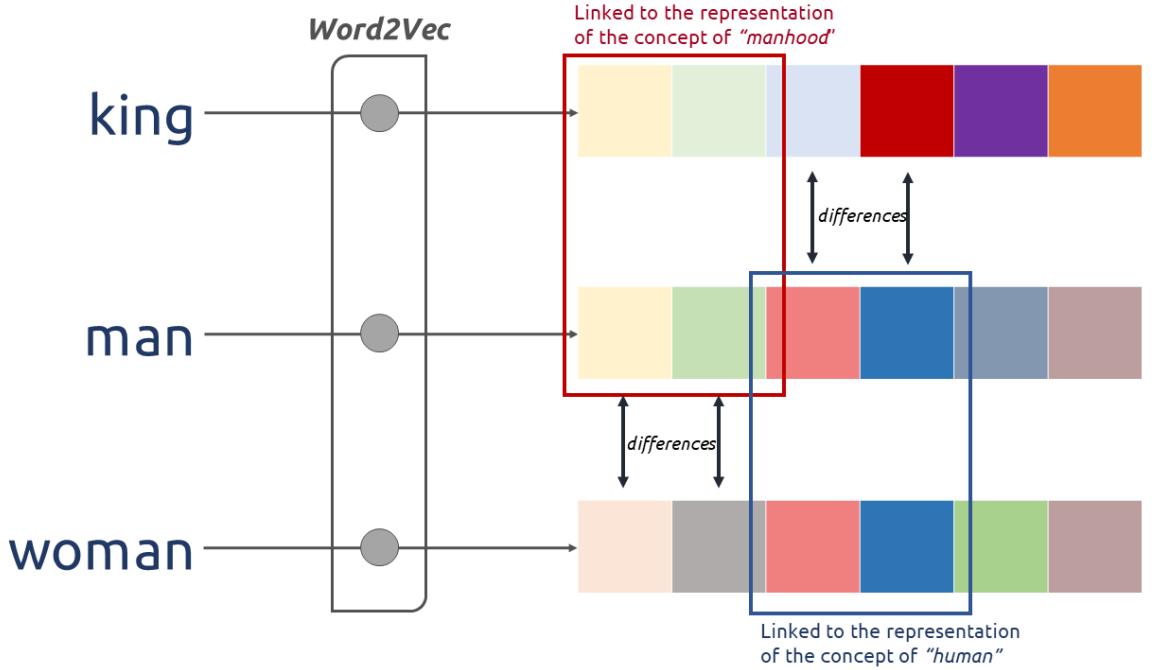


FIGURE 2.2: Word2Vec explicative graphic

These 6-sized vector representation of the words “king”, “man”, “woman” display how words with similarities will have some of their values to be closer to each other, as we see for the first two squares of “king” and “man”: they appear very similar with each other while being dissimilar with the values of “woman”. We could then suppose that these two values hold information about the shared concept of “*manhood*” and could be interchangeable in some specific context.

Such representation helps to perform word algebra where the addition and subtraction of words should lead to values similar to the encoding of other words. For instance, given that we have the following semantic equations:

- man + royal = king
- woman + royal = queen

We can deduct the following operations as true for word2vec vectors:

- $\text{word2vec}(\text{king}) - \text{word2vec}(\text{man}) \approx \text{word2vec}(\text{royal})$
- $\text{word2vec}(\text{woman}) + \text{word2vec}(\text{king}) - \text{word2vec}(\text{man}) \approx \text{word2vec}(\text{queen})$

### 2.2.2.3 Word2Vec details

The main concept behind the Word2Vec is pretty straightforward: it is an artificial neural network<sup>3</sup> with two layers that takes as an input a text corpus and outputs a set of vectors, where each vector represent each word of the corpus. The goal of Word2Vec is **not** to process text in the same way as a deep neural network would but rather to transform text into a numerical form that would then be understandable by a Deep Neural Network if needed.

**Vocabulary** Before using and training a Word2Vec model for our word embedding task, we need to settle ourselves with a vocabulary of unique words. This is the first limitation of the Word2Vec model but is a necessary step before starting any processing. For that intent, we need to list every word that appears in our training documents. For the rest of the explanation, we will consider a vocabulary of 1,000 unique words. These words should be sorted in a given way (alphabetical is preferred) for the developer comfort. While this sorting property has little to no impact on the algorithm performances, the size of vocabulary could play a crucial role. In [Christophe et al. \[2017\]](#), authors have benchmarked vocabulary of size from 10 to 300 to see the impact of the Word2Vec embedding on the overall system accuracy.

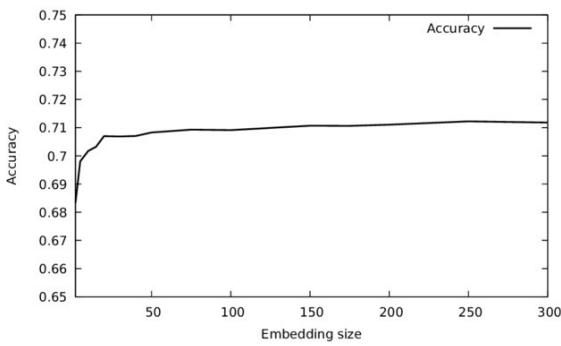


FIGURE 2.3: Influence of vocabulary size on system accuracy [Christophe et al. \[2017\]](#)

<sup>3</sup>variants of the Word2Vec exist, we are focusing on the skip-gram model

Their result, displayed in Figure 2.3, is encouraging, showing that a word embedding system with lower dimensions for the embedding does not have a necessarily bad impact on the overall system performance. However, not a lot of studies have been conducted on this matter. Indeed, when designing an NLP application using Word Embedding, it is extremely rare to retrain from scratch a Word2Vec model: most of the time, a model already trained, using an exhaustive vocabulary, is used.

**Model** Once the vocabulary has been established, it is time to design the model that will be used to compute the embedding. Multiple models can be used with Word2Vec but we will focus on the most popular one: Skip-Gram (cf Figure 2.4).

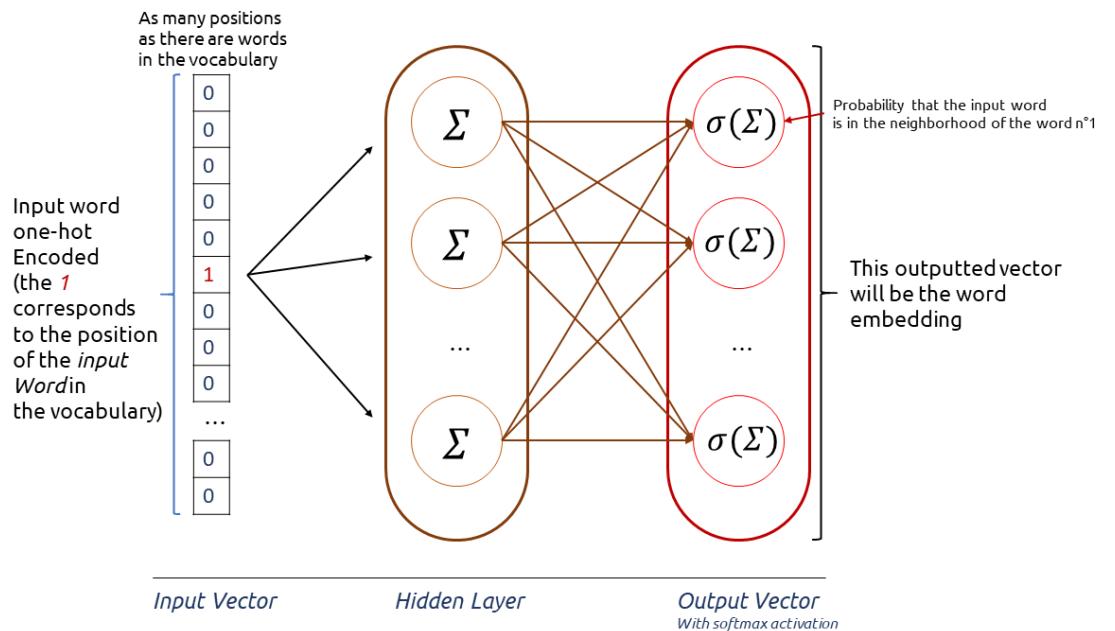


FIGURE 2.4: Graphical representation of the Skip-Gram model for the Word2Vec embedding

It takes as an input the word to be embedded transformed as a one-hot vector (a vector of the same size than the vocabulary filled with 0 except in one cell corresponding to the index of the said word in the vocabulary). This word is then passed into the first hidden layer of the network which is then passed to the output layer of the network, performing a softmax operation to output probabilities. The output vector is then a list of probabilities giving the probabilities that the input word and every other word of the vocabulary may appear in a similar context.

However, it is **crucial** to remember that main utility of the Word2Vec model is to

transform a word into a vector of values: the training context we provide the model with (i.e to output probabilities of how likely it is that words will be close to each other in a sentence) is just a **pretext** for it to extract information from the training documents. This training context can be considered as a "*Fake Task*"

For example, if we have the following two sentences in our training set:

- “Today is sunny, it is a *good* day”
- “Today is sunny, it is a *great* day”

The output of this Word2Vec model of both “*good*” and “*great*” should be very close to each other as these words appear in similar contexts.

## 2.3 Machine Learning concepts

Before jumping into Machine Learning for Similarity detection tasks, we will first introduce elementary and universal notions of Machine Learning such as the concept of a dataset, a training procedure or evaluation. This specific vocabulary is crucial to be well understood before diving into more complex notions, where a lot is considered known and understood.

### 2.3.1 What is Machine Learning ?

Machine Learning is a way of coding an algorithm so that it deducts domain-specific knowledge from data: in other words, it automates the building of n analytical model by extracting information from a dataset.

In this way, “*machine learning approach substitutes the step of acquiring domain knowledge with the potentially easier task of collecting a sufficiently large number of examples of desired behaviour for the algorithm interest.*” [Simeone, 2018]. As explained here, Machine Learning can be seen as shifting a problem from mastering a domain knowledge to collecting enough relevant data points of this domain to approximate its knowledge through statistical training.

Objectives of a Machine Learning algorithm can be numerous, but most of the time, it will either be to give a class to a piece of data or to predict a value given an observation.

The former is called “*classification*” while the latter is “*regression*”. These being different objectives, they will not have the same Machine Learning algorithms to be trained. However, training requires data, and in Machine Learning, this data is called a “*dataset*”.

### 2.3.2 Dataset

Any group of data that has coherently been collected can be considered a dataset. In other words, a dataset can be seen as a countable agglomeration of individual pieces of data, which are, most of the time, samples of observation of the real world. However, in some situations, data has to be synthetically generated [Nikolaus et al., 2018].

As said before, a dataset needs to be coherent: individual pieces of data need to have as many things in common (whether it is about format or content) to be used as a whole. A dataset is not only used in Machine Learning but also in Statistics (where it is often called a *sample* from a *population*) or in Data Analytics, where statistics are computed to give better insights on what is hidden in the data (also known as Data Mining).

However, most datasets come uncleaned or unprepared. Data Preparation is an important part of the Machine Learning pipeline.

### 2.3.3 Data Preparation in Machine Learning

In every computer vision task, an image (and by extension, a whole dataset) usually goes through the same processing pipeline (cf. Figure 2.5):

Each of these processing steps is crucial to achieve good performance and any weak link would compromise the whole chain. We will now detail the basis of each of these steps to come up with a stronger background about these notions and to better visualise the process as a whole.

We will detail every step in the Data Preparation process and will then illustrate both the Features Extraction & Feature Classification tasks using the Similarity Learning concept.

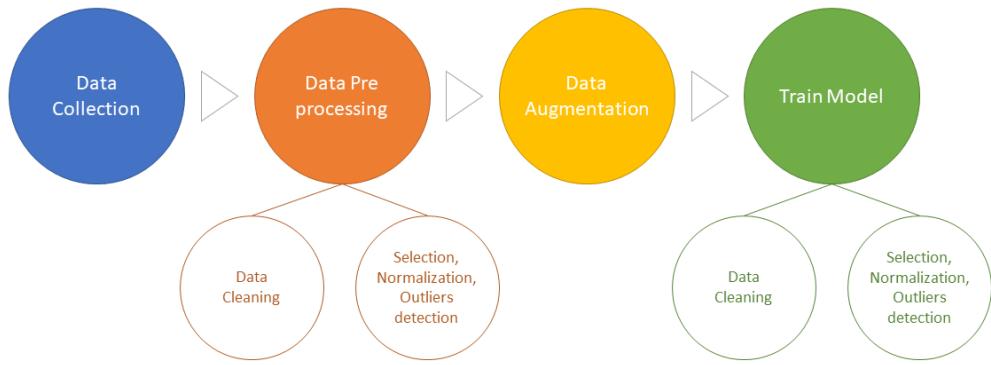


FIGURE 2.5: Graphical representation of a ML Pipeline in Computer Vision

### 2.3.3.1 Data Collection

Creating a dataset or choosing a dataset to train with for a task is one, if not the most important step of the whole process. Multiple parameters come into play when creating a dataset:

- Size of the dataset: quality of the size of a dataset is relative. For some tasks, it may be required to have thousands of elements while for others, a few hundreds may be enough.
- Diversity of dataset: a dataset is diverse if it contains a lot of classes (e.g. 21,841 classes for ImageNet [Russakovsky et al. \[2015\]](#));
- Genericity of the dataset: popular datasets such as ImageNet [\[Russakovsky et al., 2015\]](#) have a lot of tasks to be performed (classification, segmentation,...) and can be used for a lot of different problems;

### 2.3.4 Evaluating a classification algorithm

Evaluating the performance of a Machine Learning algorithm quantitatively is a very instinctive process: we test our algorithm on unseen data to see how well it generalises.

We can then compare multiple algorithms and/or datasets between each other by comparing their “*metrics*”. For some metrics, the higher the better, while for others, it will be the lower the better, but in every case “*performance metrics (error measures) are vital components of the evaluation frameworks*” [Alexei, 2018].

Considering a binary problem (i.e two outcomes, both being mutually exclusive) where one outcome is “Yes”, the other being “No”:

- do we want to always find every “No”, while taking the risk of mistaking some “Yes-es” for “No-es” ?
- Or oppositely, do we want to be sure that when a “No” is detected, it is almost impossible that it is actually a “Yes”, risking to miss some “No-es” because the risk of mistaking them was too high ?

Indeed, as seen above, a classification problem is not easy to evaluate and there is no universally better way to evaluate an algorithm, it only depends on the application domain.

The basis for evaluating classification problem is to count how many items where misclassified. A quick visualisation of this is through the confusion matrix:

|                 |                 | Classified Positive   | Classified Negative  |  |
|-----------------|-----------------|---|--|--|
|                 |                 | Considering a binary classification problem, the classified positives are any item whose predicted class is classified as positive (e.g. Yes) | Considering a binary classification problem, the classified positives are any item whose predicted class is classified as negative (e.g. No) |  |
| Actual Positive | Actual Positive | True Positive (TP)<br>Any item whose true class is positive correctly classified as positive  | False Negative (FN)<br>Any item whose true class is positive incorrectly classified as negative  |  |
|                 | Actual Negative | False Positive (FP)<br>Any item whose true class is negative incorrectly classified as positive   | True Negative (TN)<br>Any item whose true class is negative correctly classified as negative   |  |
|                 |                 |   |  |  |

|  |   |    |    |
|--|---|----|----|
|  |   | Y  | N  |
|  | Y | 90 | 10 |
|  | N | 5  | 5  |

FIGURE 2.6: Presentation of the confusion matrix

In this confusion matrix, one may think that we have a very well-performing algorithm, where 90 out of 110 predictions are correct. Even better, 90 of the *positive* items, out of a 100, are classified as *positive*. However, when looking at the *negative* items, we realise that only 50% of the *negative* items are correctly classified. In some situations, predicting the rarest class (here the negative class) is as important if not more important

than predicting the most common class (here the positive class).

For this matter, more elaborate metrics were created, on the basis on the confusion matrix, to have a large panel of performance metrics to compare two algorithms.

| Metric        | Formula                                     | Description   | Value  |
|---------------|---|---|--|
| Accuracy (A)  | $A = \frac{TP + TN}{\text{Total}}$          | The accuracy is the most basic metric. It consists in the sum of the diagonals of the confusion matrix and the division of it by the total number of items. It can be wrongly interpreted when some classes are rare: if they are misclassified, they will not weight in the calculation of the accuracy for it to show anything. | $A = \frac{90 + 5}{110}$   |
| Recall (R)    | $R = \frac{TP}{\text{Actual Positive}}$     | The recall is a more useful measure: it counts, from every positive item, how many of them were actually counted as positive  | $R = \frac{90}{100}$   |
| Precision (P) | $P = \frac{TP}{\text{Classified Positive}}$ | The precision measure usually comes alongside of the recall to describe how many of the predicted positive are actual positive.   | $P = \frac{90}{95}$  |
| F1 Score (FS) | $FS = 2 * \frac{R * P}{R + P}$              | Combining both the recall and precision metrics, the F1 score can be used to try to optimise both metrics as much as possible. In other words, the higher the F1 score, the better both Recall and Precision are.   | $FS = 2 * \frac{\frac{90}{100} * \frac{90}{95}}{\frac{90}{100} + \frac{90}{95}}$ |

TABLE 2.1: Description of other metrics

Depending on the situation and the task, different metrics may be useful and can be interpreted differently.

## 2.4 Deep Similarity Learning

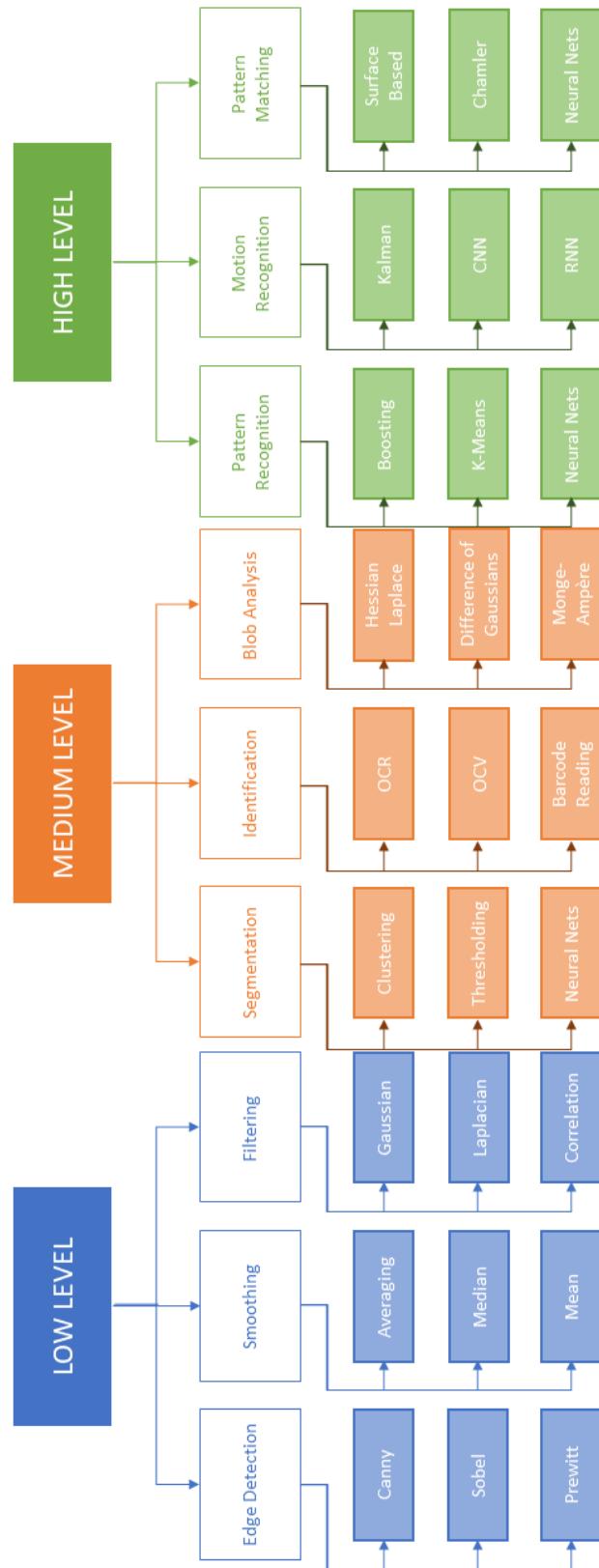


FIGURE 2.7: Taxonomy of Computer Vision

The following taxonomy (cf. Figure 2.7) shows the prominent place Machine Learning has taken in Computer Vision task over the past 20 years, and this since the revolutionary LeNet paper [LeCun et al., 1998] in 1998 that completely changed the landscape of OCR (Optical Character Recognition) techniques by creating one of the first version of Convolutional Neural Networks to recognise handwritten digits.

From this taxonomy, we can note 5 main types of application where Machine Learning seems to excel:

- Image Classification (Classifying an image based on its content, also known as *Pattern Recognition*)
- Object Detection (Detecting the position of an object, if present, in an image; also known as *Pattern Matching*)
- Object tracking (Tracking an object in a video)
- Semantic Segmentation (Give a class to each pixel of an image to create a *mask*)
- Instance Segmentation (Same as Semantic Segmentation but discriminate each instance of the same class)

## 2.4.1 Deep Learning: the new generation of statistical models

### 2.4.1.1 Deep Learning introduction

One of the most prevalent types of model in nowadays Machine Learning is the “Deep Learning” model [Wehle, 2017]: a Deep Learning model is just a “*Deep Neural Network*”, which could be defined as a Neural Network with more than 3 layers (the exact amount is debatable and there are multiple schools of thought about this).

This leads to the main difference with “*usual*” Machine Learning as in Deep Learning, the latter can approximate more complex functions the deeper and deeper they get. However, the concept of Neural Networks themselves may remain confusing.

**Neural Networks, a high-end learning model** In the simplest of ways, a Neural Networks can be seen as a network accumulating, under the form of layers, an ensemble of Perceptrons [Rosenblatt., 1958]. Taken alone, a perceptron can be seen as a regression model, either logistic or linear, depending on the activation function used on its output. The “*Perceptron*” was invented in 1958 as new supervised learning algorithm that can be used either as a classifier or a regressor. In its original form, it was used as a binary classifier: separate data in two different classes. For this intent, it learnt a linear boundary represented as a linear function to separate data: objects of the same classes would be found on the same side of the boundary.

### 2.4.1.2 Deep Learning Models: a quick overview

As seen before, Deep Learning models are Deep Neural Networks [Liu et al., 2016]. A Deep Neural Network can be, for most of them, categorised in of these 3 types:

- Dense Neural Networks, used for classification or regression. They expect the input to already be full of features;
- Convolutional Neural Networks, also known as CNNs, help to capture spatial information in unstructured data (like images or sound). They have the particularity of being composed of convolutional layers that extract features from the input data;

- Recurrent Neural Networks, also known as RNNs, help to capture temporal information in unstructured data (like time series or text). They have the particularity of being recurrent: the same weights are used in every layer of the network.

**Dense Neural Networks** Dense Neural Networks are simply the “*basic*” artificial neural networks described above with a certain number of layers (the exact amount is debatable, but more than 2 hidden layers usually makes a model that can be considered as a Deep Neural Network). The deeper a network is, the more complex the model becomes (cf. Figure 2.8).

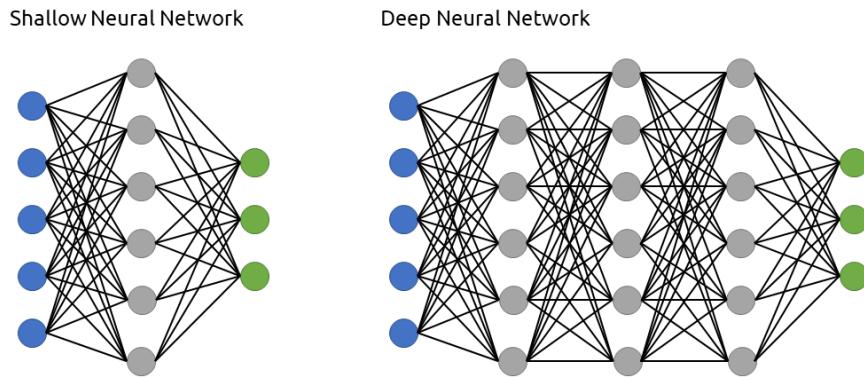


FIGURE 2.8: Comparison of a Deep and a Shallow Artificial Neural Network

A deeper model is not always preferred to a shallow one as the former have more tendencies to overfit the training data and can, then, perform worse than a simpler model. For any task, a more complex solution should be kept if and only if it **significantly** improves the system performance. If the said increase is insignificant or not safely interpretable, then the simpler model should be kept.

**Convolutional Neural Networks** An issue existing with *plain* dense neural networks is that it expects its inputs to be dense in features and information. This may be the case with structured data (like entries of a .csv files) but is **not** the case for what is called “*unstructured*” data which refers to images or text data. Indeed, as raw

data, they are hardly exploitable by a Dense Neural Network. Hence, to extract features from such data, a special kind of neural networks called Convolutional Neural Network [O'Shea and Nash, 2015] (or CNN) are used.

Inspired at first by the neocognitron [Fukushima, 1980]: presented as a “*self-organising neural network model*” that would resist to features’ deformation like local shifts of a pattern. This particular ability is what inspired the original Convolutional Neural network [Lecun et al., 2000].

**Convolutional Layers** A Convolutional Neural Network consists of the use of convolutional filters that can extract multidimensional features. These features are extracted automatically by the concatenation of multiple convolutional layers on top of each other: the deeper we are in the layers, the more complicated the extracted features are.

The convolution process is described in Figure 2.9: a filter is applied as a sliding window on top of an image and then do an element-wise multiplication.

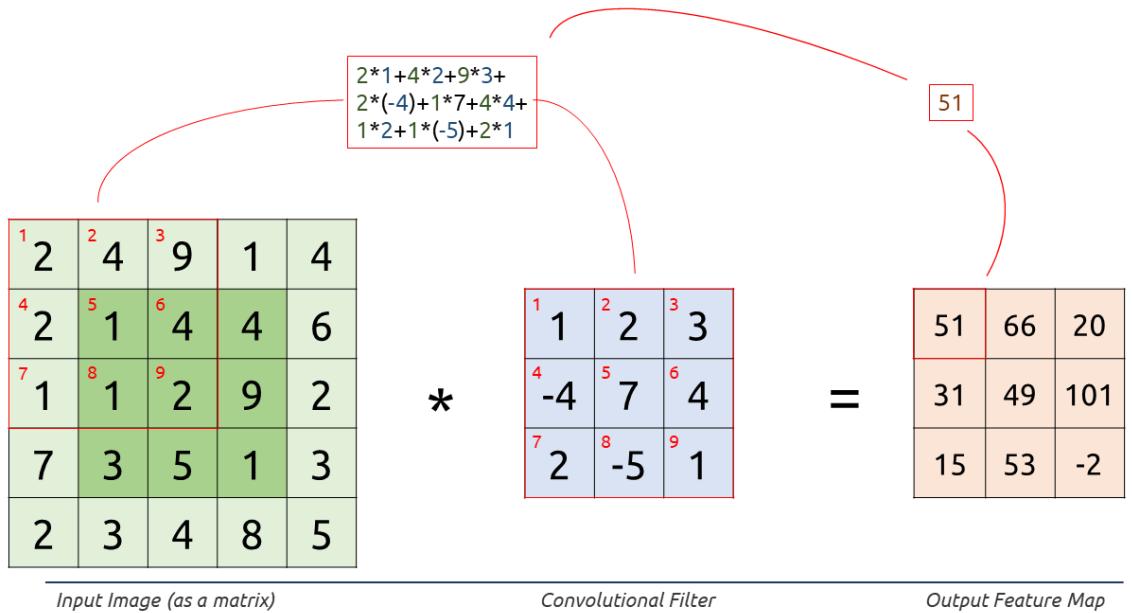


FIGURE 2.9: Simplified convolution example

Strictly defined, the convolution process will be the following.

Let  $A$  be a image input of size  $(n, m)$  where  $n, m \in \mathbb{N}^*$ ,  $F$  be a filter of size  $(s, s)$ <sup>4</sup> where

<sup>4</sup>Filters are always of square size

$s \in \mathbb{N}^*$ . We then define the convolution product of  $s$  applied to  $A$  as an output matrix  $O$  of size  $(s, s)$  such that:

$$O(x, y) = \sum_{i=0}^{s^2-1} A(x - 1 + i \mod s, y - 1 + \lfloor i/s \rfloor) * F(i \mod s, \lfloor i/s \rfloor)$$

**Pooling Layers** In between convolutional layers, most CNNs use “*Pooling layers*” which are here to “*postprocess*” the data after a convolution filter has been applied. There exist multiple Pooling layers, with different benefits:

- Average Pooling Layer (cf. Figure 2.10, where a 2x2 pooling procedure is displayed): applying an averaging operation over a region of multiple cells of a Feature Map (i.e the output of a convolutional layer). It has the property of smoothing the extracted features.

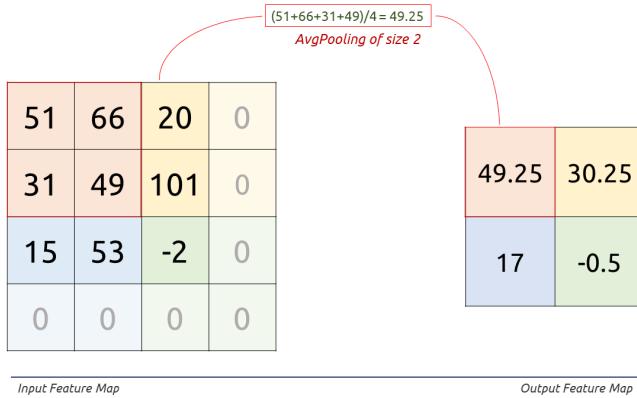


FIGURE 2.10: Average Pooling example

- Max Pooling Layer [Yamaguchi et al., 1990] [Ciresan et al., 2012] (cf. Figure 2.11, where a 2-sized pooling procedure is displayed): applying a max operation over a region of multiple cells of a Feature Map. It has the property to sharpen the extracted features (only keep the most relevant and activated features). However, this analysis is empirical. In a theoretical way, pooling layers have yet to be fully understood.

Pooling Layers will also induce dimensionality reduction of the input data. The degree of shrinkage depends on the size parameter of the pooling procedure. In our figures (cf. Figure 2.10), we apply a pooling size of 2, which induce an approximate dimensionality

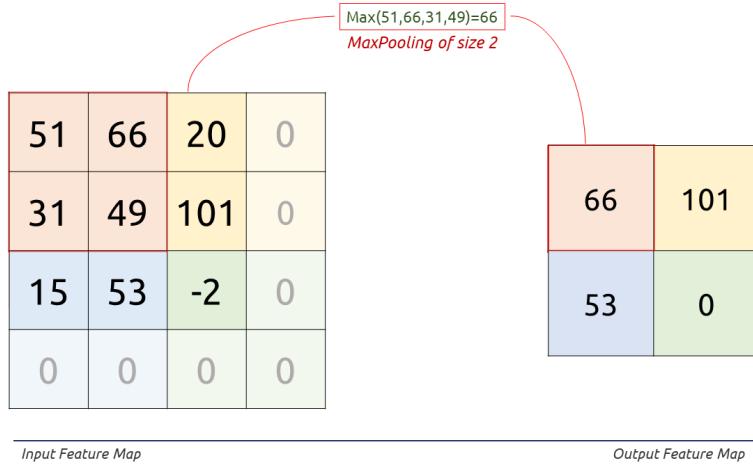


FIGURE 2.11: Max Pooling example

reduction of 2 for vertical and horizontal dimensions: the pooling algorithm will split the matrix in distinct squares of size 2 and apply, for each of them, the appropriate operation. Hence, each of these 2x2 square results in a 1x1 square, hence the reduction of dimensions. However, if the size of the feature map is not dividable by the size of the pooling process (respectively 3 and 2 here), a process called “padding” is done and 0 are added all around the feature map until its size can be pooled.

## Recurrent Neural Networks

**RNN Overview** Reviewed in details by [Sherstinsky \[2018\]](#), recurrent neural networks are a special kind of neural network where the computation flow of a specific input can be represented as a temporal sequence. For this reason, RNNs are the best neural network architecture to manipulate sequences of any kind like:

- Text;
- Speech;
- Time Series.

As explained formally by [Zachary Chase and John \[2015\]](#), “*Recurrent neural networks (RNNs) are connectionist models that capture the dynamics of sequences via cycles in the network of node*”. As displayed in Figure 2.12, these cycles exist between time steps, to provide time-dependent information: this network can be seen as a common deep

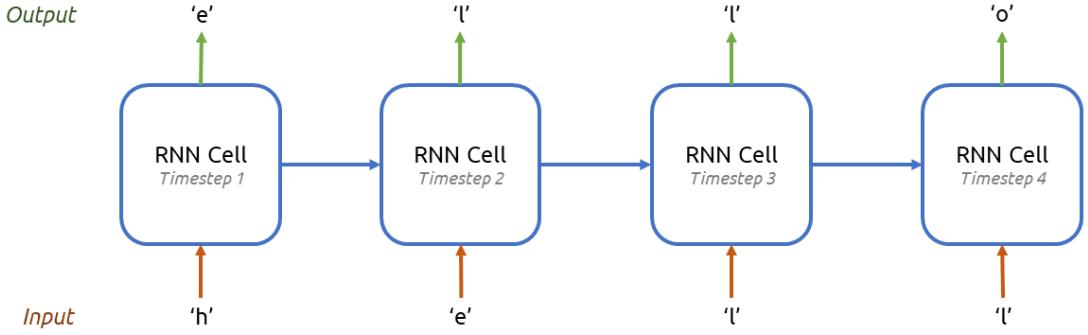


FIGURE 2.12: Simplified representation of an RNN architecture predicting the next letter in an input word splitted as a sequence of letters

neural network with shared weights across the different time steps.

To detail the calculations involved in the functioning of an RNN, let's first have a look at a detailed representation of the variable within an RNN and where they are “transmitted” (cf. Figure 2.13). Given a time-step  $t$ , the computations involved in

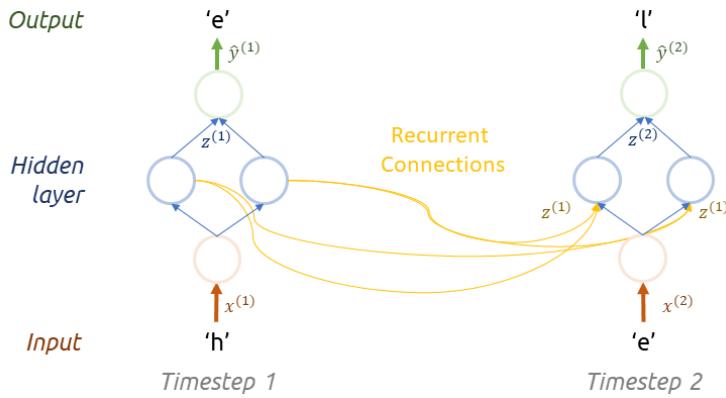


FIGURE 2.13: Details of structure of an RNN network

generating the output are the following:

$$z^{(t)} = \sigma(W^x x^{(t)} + W^z z^{(t-1)} + b_h) \text{ and } y^{(t)} = \text{softmax}(W^y h^{(t)} + b_y) \text{ where :}$$

- $W^x$ ,  $W^h$  and  $W^y$  are, respectively, the weights of the neural networks' connection between **the input and the hidden layer**, between **the hidden layers of two timesteps** and between **the hidden layer and the output**;
- $z^{(t)}$  is the activation of the hidden layers at timestep  $t$ ;
- $b_h$ ,  $b_y$  are the biases of, respectively, the connections between **the input and the hidden layer** and between **the hidden layer and the output**. They allow neurons to be able to learn offset if needed: if learning a constant value is optimal,

they will then set the neuron weight to 0 and the bias to this value. If the bias is useless, it will be kept to 0 as well.

- $\sigma$  is the activation function of the hidden layer while  $\text{softmax}$  is the activation function of the output layer.

In 1997, [Sepp and Jürgen \[1997\]](#) introduced an evolved concept of Recurrent Neural Networks: the LSTM (standing for Long-Short Term Memory).

**LSTM, a particular type of Recurrent Neural Network** The LSTM [[Sepp and Jürgen, 1997](#)], from the family of the Recurrent Neural Networks (or the *RNNs*), has been the most popular model of RNN in sequence processing. Taking the same concept of having a shared RNN Cell between each timestep of the input sequence, the difference of the LSTM is in the complexity of said cell (cf. Figure 2.14). An LSTM cell use a

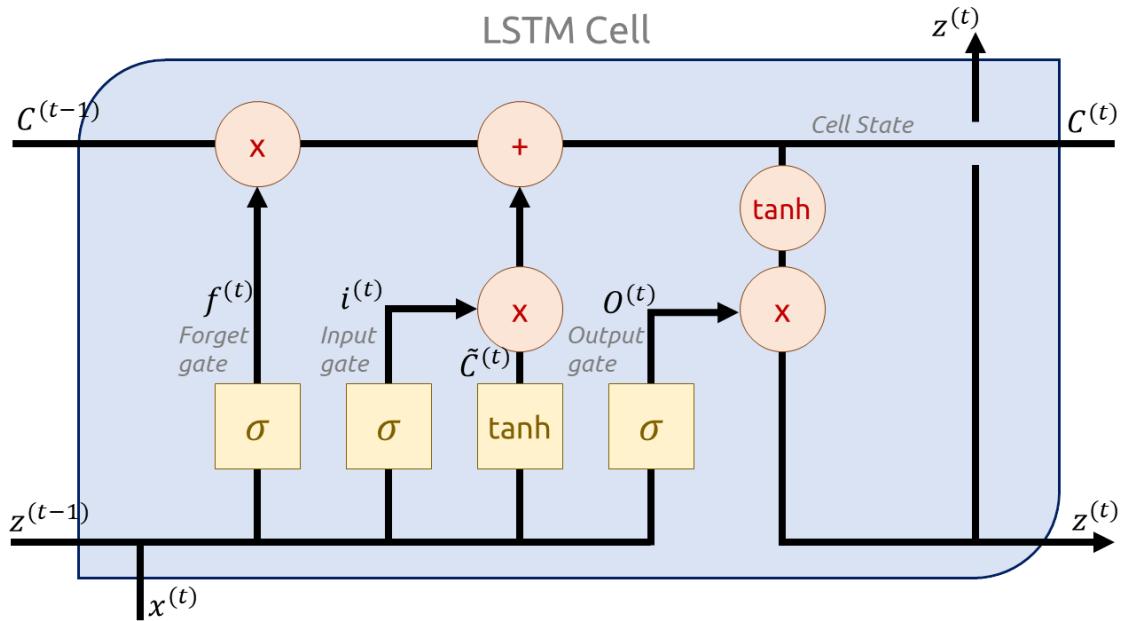


FIGURE 2.14: Graphical representation of an LSTM cell

concept of “gates” [[Sepp and Jürgen, 1997](#)]. Gates is just a special name for a data gateway. An LSTM holds 3 of these gates, all with a different objective:

- The **Forget Gate**: As information from a timestep to another is passed within from one LSTM to the next <sup>5</sup>, it is the LSTM responsibility of the LSTM to

---

<sup>5</sup>it always is the same LSTM, it is just easier to conceptualise an RNN as a series of cells with shared weights

forget this information or not. A good example of an application of this concept is in language processing: plurals can have an impact pretty late in a sentence (e.g These biscuits, which I love, are out of stock). Hence, to capture this kind of temporal dependency, an LSTM needs to know which information from former timesteps to keep (e.g subject was plural) and which to forget (e.g after the verb is conjugated, the plural information *may* become useless);

- The **Input Gate**: The input gate is taking both the former information ( $z^{(t-1)}$ ) and the new information ( $x^{(t)}$ ) to process it and send it to the cell state calculation process. We will talk about the notion of cell state later on.
- The **Output Gate**: this gate decides the output of the network. This also has an impact on the next prediction as the output is passed to the LSTM of the next timestep. However, as one may have noticed, an LSTM unit passes information through two canals ( $C^{(t)}$  and  $z^{(t)}$ ) oppositely to the simpler RNN that only uses the  $z^{(t)}$ .

These transmitted information are called “*state*” [Sepp and Jürgen, 1997]:

- The **Hidden State** ( $z^{(t)}$ ): the hidden state, passed between timesteps, is just a share of the output information from one LSTM cell to the next. It can be important for a newer LSTM cell to know exactly what was outputted by its predecessor. However, when the same information needs to be transmitted through multiple LSTMs, the hidden state is not usable anymore. We would rather use the cell state.
- The **Cell State** ( $C^{(t)}$ ) is the way, for a specific LSTM, to pass information to future cells. For a cell to renew the cell state, it generates, at the input gate, what is called a “*Candidate Cell State*” (written  $\tilde{C}^{(t)}$ ) that could replace the former cell state.

With all these tools of passing information through timesteps, LSTMs capture in a better way long-range relationships between multiple timesteps of data. This is why they are used as feature extractors and auto-encoders for similarity learning with sequential data (here textual data).

### 2.4.1.3 Training a Deep Learning model: the Back-Propagation algorithm

The default pseudo-algorithm for backpropagation is presented in Algorithm 2.

---

**Algorithm 1** Backpropagation algorithm

---

**Input:**

X: Training Data set ( $m$  examples of size  $n$ )

Y: Labels for X

W: weights of the model

```

1: procedure TRAIN( $X, Y, W$ )      ▷ The generic backpropagation pseudo-algorithm
2:   Initialize all weights  $w$  to small random numbers
3:   for  $i=1$  to  $m$  do           ▷ Looping over all training examples
4:      $a^i \leftarrow feedforward(x^i, W)$ 
5:      $e^i \leftarrow loss(a^i, y^i)$           ▷ We calculate the error
6:     for  $w$  in  $W$  do
7:        $update(w, e^i)$       ▷ We update the weights accordingly to this error
8:     end for
9:   end for
10:  end procedure
```

---

Backpropagation consists in simple iterative operations where, for every element of the input dataset, we compute a forward pass through the network of this element, calculate the error using a loss function and then update the network's weights according to the error outputted by the loss function. As we can observe, the different flavours of Deep Learning algorithms find, mostly, their differences within these 3 “parameters”.

- *feedforward(...)*: how the model is built (number/type of layers, neurons, activation functions, layer connections...).
- *loss(...)*: how the error is calculated. Depending on the problem (e.g. classification, regression..), we would like to calculate the error differently. The goal of training a Deep Neural Network is, usually, to minimise this loss function.
- *update(...)*: how the model's weights are updated, which update rule is used, which optimiser is used (e.g. ADAM [[Kingma and Ba, 2014](#)]).

Hence, advances in Deep Learning are usually improvements in one of these 3 parameters. And this is also the case for the field of Similarity Learning.

### 2.4.2 Similarity Learning

Similarity recognition is a task performed daily by an incredible amount of software. The most notable is the face recognition algorithms used by nowadays modern phones to identify the user. Another example is the iris recognition algorithm. Explored by [Li et al. \[2017\]](#) as a potential solution to improve the protection of mobile devices, Iris Recognition has been proven to be very efficient when using the right techniques to tackle challenges such as low resolution and noisy data.

In the context of images or text, we are struggling with the “*curse of dimensionality*”. Indeed, to detect similarity between low-dimensional data records is easy (e.g. comparing two persons using their height and weight can be seen as “*an easy task*” as it falls under a 2-dimensional configuration). But images are not 2-dimensional: for example, a colour image of size 256 by 256 is a 196,608-dimensional data record. It then becomes very hard to process it “*as is*” for similarity task. We need to reduce the initial image to a vector of features. Hence, to measure the similarity between images, we would just have to compare their feature vector. This is the same concept idea for text data.

Hence, Deep Learning becomes useful as a powerful feature extractor for our data. It will then create a latent representation of this input data in the form of a feature vector. This use of Deep Learning to create vectors of data and compute distances between vectors is also known as “*Deep Embedding Learning*” [[Duan et al., 2019](#)].

Formally, we can reduce deep embedding learning as being:

$$sim(X_1, X_2; W) = \begin{cases} 1 & \text{if } \|f(X_1; W) - f(X_2; W)\| < \alpha \\ 0 & \text{if } \|f(X_1; W) - f(X_2; W)\| \geq \alpha \end{cases} \quad (2.1)$$

where :

- $W$  represents the weights of the Deep Learning model;
- $X_1$  and  $X_2$  are the model’s inputs;
- $f(X_i; W)$  represents the embedding function that transforms the input data into a feature vector.

Our objective here is to train our model to get the “*most informative data embedding*” by learning  $W$  so that the distance metric can output a similarity score for the two

inputs.

Variations in literature can then take multiple forms:

- The design of the neural network architecture
- The weight-update method, which varies depending on the loss
- The sampling method: which heuristic to use to choose which data to use to train the network.

We will now explore these 3 different possibilities by providing examples of the most noticeable improvements in the field made by exploring these concepts.

#### 2.4.2.1 Design of a Deep Neural Network for Similarity Learning: the Siamese Neural Network

One of the most relevant paper on the subject of using a DNN for learning embeddings that will then be used to detect similarities between data samples is *Dimensionality Reduction by Learning an Invariant Mapping* [Hadsell et al., 2006]. In this paper, the authors present a rightfully named method: the “*Dimensionality Reduction by Learning an Invariant Mapping*” (DrLIM). The goal of this method is to be able to create a mapping reducing the input data dimensions while at the same time being invariant to “*little*” transformations of the said input data.

Formally, it would be about finding a function  $G_W : \mathbb{R}^n \rightarrow \mathbb{R}^t$ , where  $n \gg t$  so that we have:

$$\|G_W(x) - G_W(\tilde{x})\|_2 \approx 0, \forall x \in \mathbb{R}^n$$

where  $\tilde{x}$  is a transformed version of  $x$ , and  $G_W$  is the invariant mapping function, weighted by the learnt parameters  $W$ .

This method presents a “*siamese*” architecture that it defines as consisting of “two copies of the function  $G_W$  which share the same set of parameters  $W$ , and a cost module”. Each  $G_W$  function is a convolutional neural network as presented in : The authors of the DrLIM method detailed in their results how this method helped to semantically cluster the MNIST dataset: a dataset of handwritten digits (hence 10 possible classes, from 0 to 9). In the results presented in Figure 2.16, we see that before training, the images were clustered regardless of semantic, in the sense that they were classified whether they were

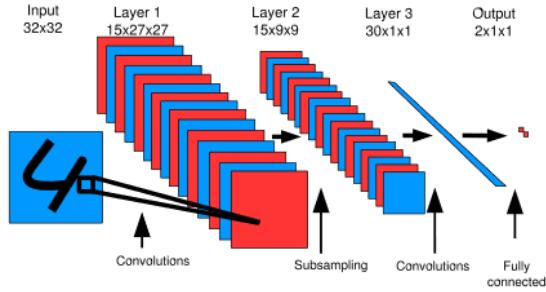


FIGURE 2.15: Architecture of the  $G_W$  mapping function (source: [Hadsell et al., 2006])

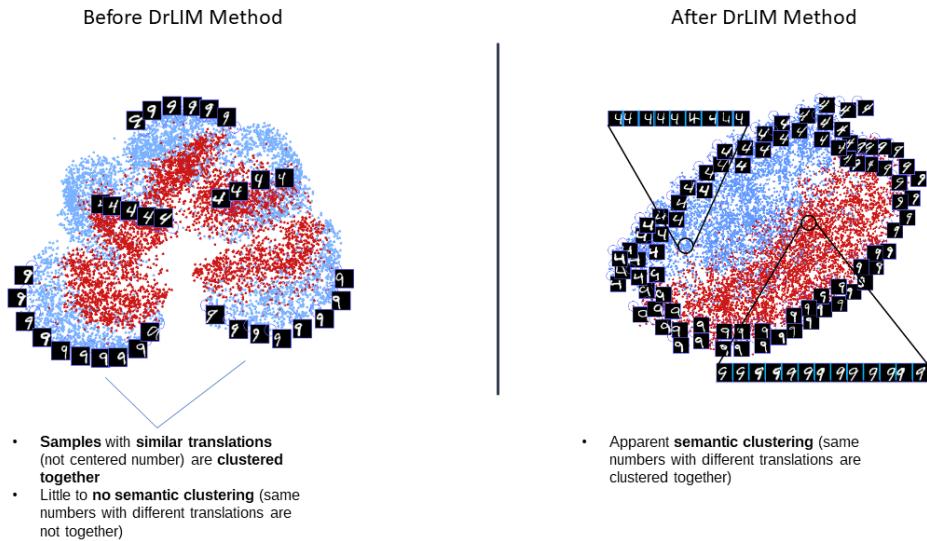


FIGURE 2.16: Results of DrLIM Method on semantic clustering of MNIST dataset (source: [Hadsell et al., 2006])

representations of 9s or 4s but only based on the symbol and its position/orientation in the image. However, when the network was trained to recognise the concept of a “9” in an image, he was then able to position the images of similar numbers closer together in the 2D space.

These promising results were the first step in the direction of other solutions implementing Siamese Neural Network. While the task could be considered as easy for the case of MNIST as a lot of samples of every number exist, the true strength of Siamese Neural Network is their ability to work with not very well distributed data where the number of classes exceeds by far the number of samples.

This application Siamese neural networks has been discussed by Gregory et al. [2015]. The authors used this model to learn the deep embedding of handwritten characters from multiple alphabets (cf. Figure 2.17). They pointed out the idea of the siamese

neural network as being particularly useful in a context where they only disposed of a couple of samples for every class of character. The siamese CNN architecture used by

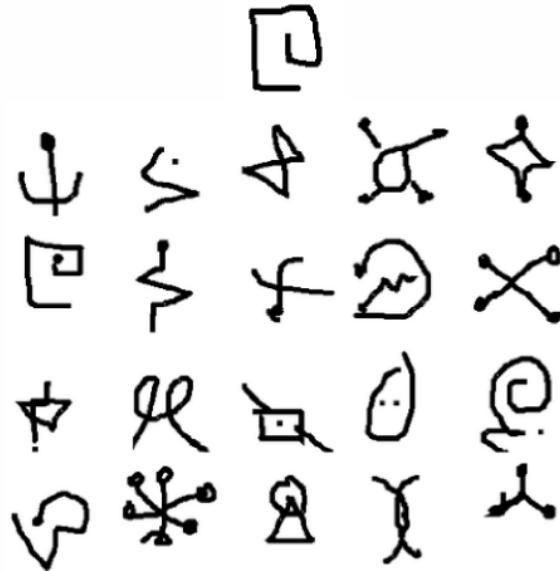


FIGURE 2.17: Samples from the Omniglot dataset (source: [Gregory et al., 2015])

the authors of this paper is shown in Figure 2.18.

This architecture duplicates the CNN network and extracts from the two images two

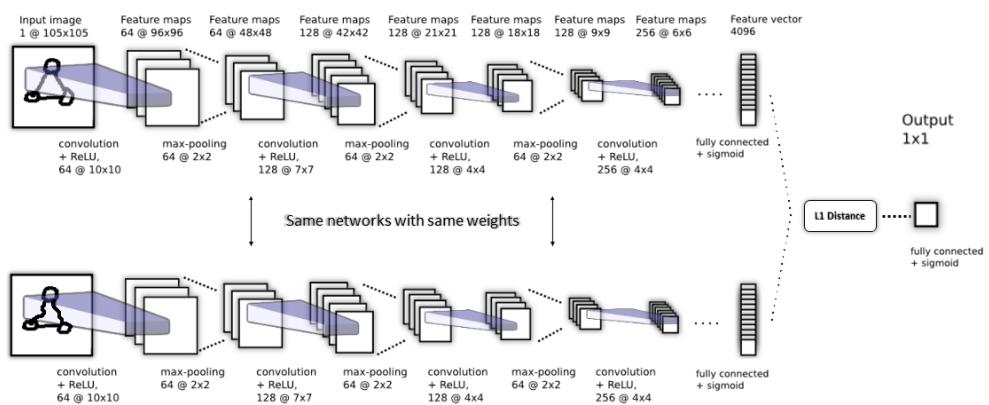


FIGURE 2.18: Convolutional Siamese architecture (source: [Gregory et al., 2015])

4096-sized vectors that are then reduced to a distance value using the  $l_1$ -norm that is then passed into a sigmoid layer to generate an activation between 0 and 1, providing a confidence score. If we relate this network to our general formula 2.1, this model would

be calculating the similarity score in the following way:

$$sim(X_1, X_2; W) = \begin{cases} 0 & \text{if } \sigma(\sum_{i=0}^{4095} |G_W(X_1)_i - G_W(X_2)_i|) < 0.5 \\ 1 & \text{if } \sigma(\sum_{i=0}^{4095} |G_W(X_1)_i - G_W(X_2)_i|) \geq 0.5 \end{cases} \quad (2.2)$$

where :

- $\sigma(x)$  is the sigmoid function defined by  $\sigma(x) = \frac{1}{1 + \exp^{-x}}$

They achieved a 92% one-shot accuracy on unseen test set which put them right behind human level, evaluated at 95.5% accuracy (cf. Figure 2.19).

| Method  | Test |
|---|------|
| <b>Humans</b>                                 | 95.5 |
| <b>Hierarchical Bayesian Program Learning</b> | 95.2 |
| <b>Affine model</b>                           | 81.8 |
| <b>Hierarchical Deep</b>                      | 65.2 |
| <b>Deep Boltzmann Machine</b>                 | 62.0 |
| <b>Simple Stroke</b>                          | 35.2 |
| <b>1-Nearest Neighbor</b>                     | 21.7 |
| <b>Siamese Neural Net</b>                     | 58.3 |
| <b>Convolutional Siamese Net</b>              | 92.0 |

FIGURE 2.19: Comparison of different networks against baselines on one-shot accuracy over the Omniglot dataset (source: [Gregory et al., 2015])

However, as seen above, other parameters are to be taken into account when improving a model. Another important being the loss function used to calculate the error and then train the network.

#### 2.4.2.2 Losses of Deep Similarity Learning

**Binary Cross-Entropy Loss** As described by Gregory et al. [2015], the problem of similarity learning can be considered as a simple binary classification task where the class  $0$  is for the similar data and the class  $1$  is for the dissimilar data. In this classic context, the authors of Gregory et al. [2015] used in their implementation a variant of the Binary Cross-Entropy:

$L(\hat{y}, y) = -(y)*\log(\hat{y}) - (1-y)*\log(1-\hat{y})$ . Minimising the loss with the backpropagation

algorithm would make the model learn to create similar embedding for similar data inputs. Depending on the class of the input, the loss function will be shaped differently and push the training in a different “*direction*” (cf. Figure 2.20).

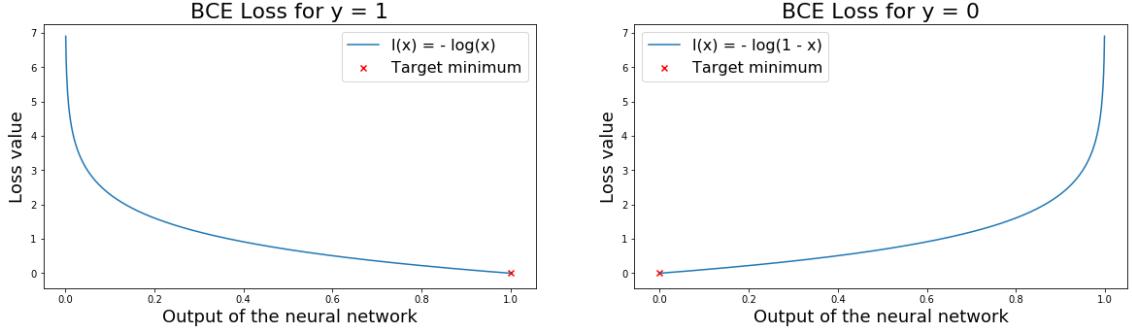


FIGURE 2.20: BCE Loss comparison

**Contrastive Loss** Initially presented by Hadsell et al. [2006], this loss introduces the concept of margin: As presented in Figure 2.21, the Contrastive Loss have this same

$$\text{loss}(d, Y) = \frac{1}{2} * Y * d^2 + (1 - Y) * \frac{1}{2} * \max(0, m - d)^2$$

Where:

- $d$  is the distance between the outputs of the encoder;
- $Y$  is the label of the model inputs (1 if *similar*, 0 if *dissimilar*);
- $m$ , the margin parameter

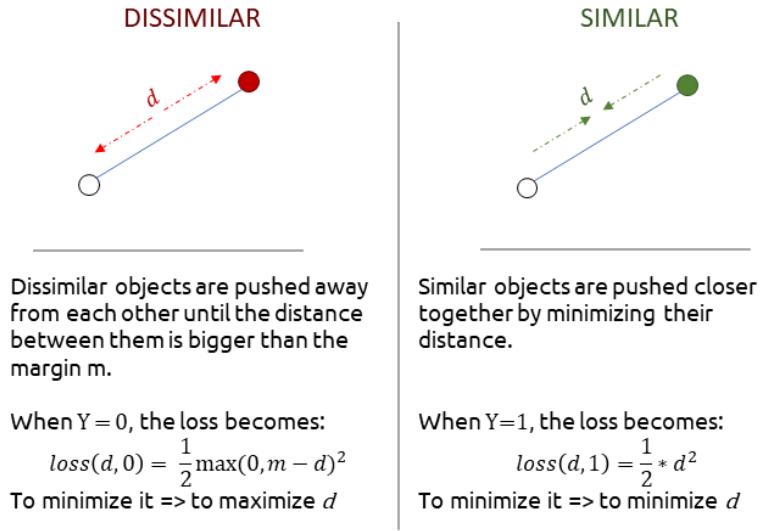


FIGURE 2.21: Contrastive Loss illustration

duality of equations as for the BCE Loss: we treat both cases of *similar* and *dissimilar* data disjointly. We try to group similar data while separating dissimilar data. For that intent, a margin parameter is defined: this margin is here so that once every dissimilar

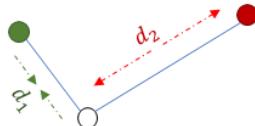
data samples' distance with other data samples is above this margin, the training of the neural network will not try to push them further away and will mainly focus on grouping closer together similar objects. However, one may argue that training on two inputs, that can be either similar or dissimilar can be seen as yet another “*greedy*” technique: indeed, by iteratively jumping from similar and dissimilar data inputs, we may end up with a back and forth training process that could be detrimental to correctly cluster the manifold. This problem exists also with the BCE Loss and has been discussed by [Schroff et al. \[2015\]](#) where they introduced the “*Triplet Loss*”.

**Triplet Loss** In this revolutionary paper, [Schroff et al. \[2015\]](#) presents a new way to train Siamese neural networks by using what is called a ”*Triplet Loss*”. They apply it to the task of facial recognition where they worked on making their model invariant to illumination and pose change. They used their trained network to transform face images into embedding of lower dimensions to that faces of same persons would have “*small*” distances while faces of different persons would have “*big*” distances, these notions of *big* and *small* being relative.

$$\text{loss}(d_1, d_2) = \max(d_1^2 - d_2^2 + m, 0)$$

Where:

- $d_1$  is the distance between two similar examples;
- $d_2$  is the distance between two dissimilar examples;
- $m$  is the margin parameter.




---

At each iteration, an **anchor** (white) is pushed closer to a **similar object** (green) while, at the same time, it is pushed away from a **dissimilar object** (red) until their distance is superior to the margin parameter.

To minimize the loss, our model will then follow two strategies at the same time:

- Minimize  $d_1^2$
- Maximize  $d_2^2$  until it is superior to  $m$  (this is defined by the max function)

FIGURE 2.22: Triplet Loss illustration

With triplet loss, the network is copied so that 3 versions of it exist, all sharing the same weights. Instead of alternatively sampling dissimilar and similar data items, [Schroff et al., 2015] detailed this new idea of sampling 3 data records at each timestep:

- The anchor sample: acting as the basis of the 3-sized input, it will be used to select the two other samples;
- The positive sample: this data sample is assumed similar to the anchor sample;
- the negative sample: this data sample is assumed dissimilar to the anchor sample (and logically to the positive sample as well).

As presented in Figure 2.22, we can see that the triplet loss is trying to work on two things at the same time: minimising the distance between the anchor and the positive sample, here represented as  $d_1$  as well as maximising the distance between the anchor and the negative sample, here represented as  $d_2$ , until it is above the specified margin  $m$ .

Using their method, Schroff et al. [2015] achieved an astounding 30% error reduction over the state-of-the-art solution of the time on the Youtube Faces DB with a 95.12% classification accuracy.

During their work, Schroff et al. [2015] pointed out the importance of creating a relevant group of inputs to make training more efficient and performant. This brings us to our last point: Data Sampling.

#### 2.4.2.3 Data Sampling: a crucial asset for Similarity Learning

In any Deep Learning tasks, not every data pieces are as hard to process as every other: some may be harder to process due to the unusual information embedding, unusual with regards to the rest of the dataset. Such data records are called “*hard samples*”. Hard samples can be objects in an unusual illumination or unusual orientation. To get a more performant Deep Learning algorithm, one needs to have a good balance of easy and hard samples so that the network can always improve itself. Hence, it sounds logical to have a sampling heuristic to optimise training time and convergence.

To draw the parallel with Similarity Learning, hard samples may either be:

- **similar** data records where current **distance** is **higher** than some **dissimilar** data records. The features used by the network to classify them as similar may be either hard to find or not yet learnt by the model. Further training by focusing on these samples may resolve this issue.
- **dissimilar** data records where current **distance** is **lower** than some **similar** data records. The network confuses the two data records as being similar, they need to be “*pushed away*” from each other by training the network to recognise their difference.

Hence, we have this idea of choosing our training samples depending on how much they can help the network to learn.

**Distance-weighted sampling** In Wu et al. [2017], the authors present a method called “*Distance-weighted sampling*”. Here, for each anchor, they “*sample uniformly according to the distance*” [Wu et al., 2017] by sampling positives using  $d$ , hence prioritising the positives with higher distances, and sampling negatives using  $d^{-1}$ , hence prioritising the negatives with lower distances.

**Mini-batch-wise hard sampling** When training a Deep Learning model, it is very common to split the full training dataset in mini-batches of  $2^n$  element where n is usually between 4 and 10, depending on the performance of the computer used for training. Cutting the full training set in mini-batches improve convergence and makes training possible for very large datasets Li et al. [2014]. Used in Schroff et al. [2015], the method of “*Mini-batch-wise hard sampling*”, also called “*Online hard-triplet mining*” will, within every mini-batch of data, find the hardest negative and positive examples for a given anchor.

This helps training as it finds the most “*interesting*” data sample for every anchor at every timestep, hence improving training. Contrary to Wu et al. [2017], this method does not sample randomly and simply select the hardest. However, the author of Schroff et al. [2015] explains that selecting hard negatives too early in training may result in bad local minima and even in a collapsed model (constantly predicting the same output). To mitigate this, they find negatives which distances are close to the distance the anchor

would have with positive samples. These samples' distances are usually still smaller than the margin parameter and would then need to be “*pushed away*”. They are called “*semi-hard negatives*”.

### 2.4.3 Related methods in Multimodal Deep Learning

As detailed at the start of this chapter, Deep Learning has been extensively used and explored as an automatic feature extraction tool. It has taken over a lot of discipline and we covered how it is now prevalent in areas such as Similarity Learning.

#### 2.4.3.1 Stacked-Autoencoder to learn multimodal embedding

In this exact context, Siamese Neural Networks seem to shine: they offer an ideal approach to Similarity Learning by using the same processing for each data record to compare, by evaluating their similarity with the data representation resulting of this processing and finally, by adapting the weights so that similar data will have similar latent representation and dissimilar data will have dissimilar latent representation.

Hence, one may think that any preprocessing pipeline would be good enough for the task, as long as it identifies objects while being invariant to change in the data. This means that methods and models built for other purposes could work in such context, as long as it generates an exploitable embedding. For example, [Silberer and Lapata \[2014\]](#) present the idea of using multimodal autoencoders to learn multimodal embedding. An autoencoder is a simple yet powerful Deep Learning model which task is to learn two functions:

- The encoder function, that we will note  $e_W : \mathbb{R}^n \rightarrow \mathbb{R}^s, n >> s$ ;
- The decoder function, that we will note  $d_W : \mathbb{R}^s \rightarrow \mathbb{R}^n$ .

The goal of  $e_W$  is to “*compress*” the input data into a latent representation of dimension  $s$  with as little information loss as possible. The goal of  $d_W$  is to reconstruct the initial data using the output of  $e_W$ . Formally speaking, an autoencoder will calculate:  $d_W(e_W(x)) = \tilde{x}$  where  $\tilde{x} \approx x$ .

In their work “*Learning Grounded Meaning Representations with Autoencoders*”, [Sil-](#)

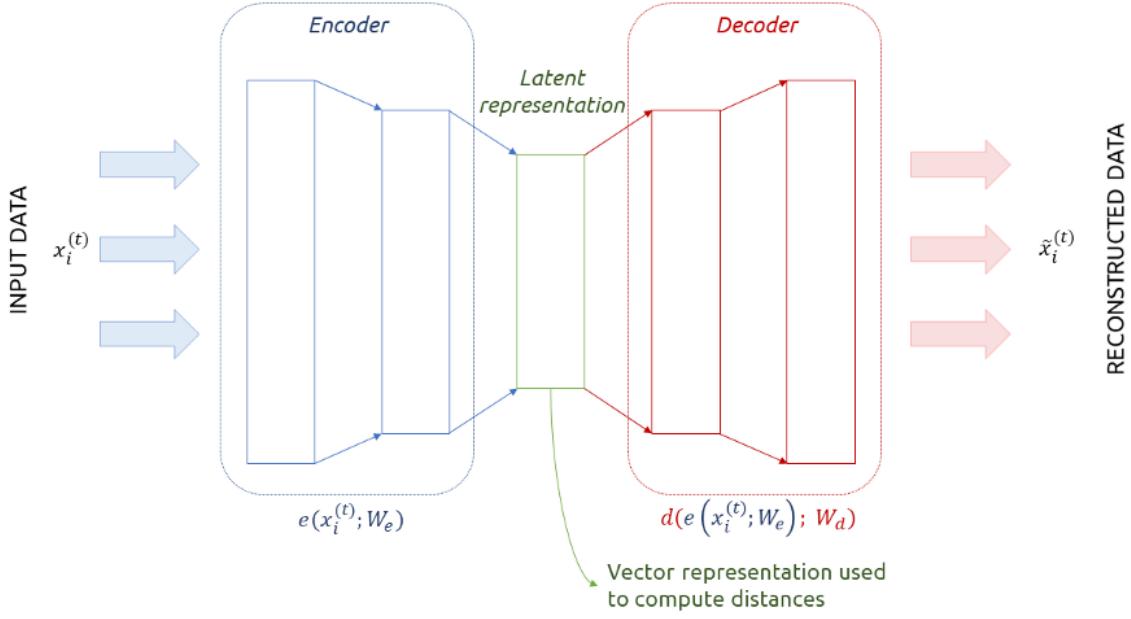


FIGURE 2.23: Autoencoder illustration

Silberer and Lapata [2014] explored a concept where they would combine the multiple modalities (text and image) into one single feature vector: In their work, they trained

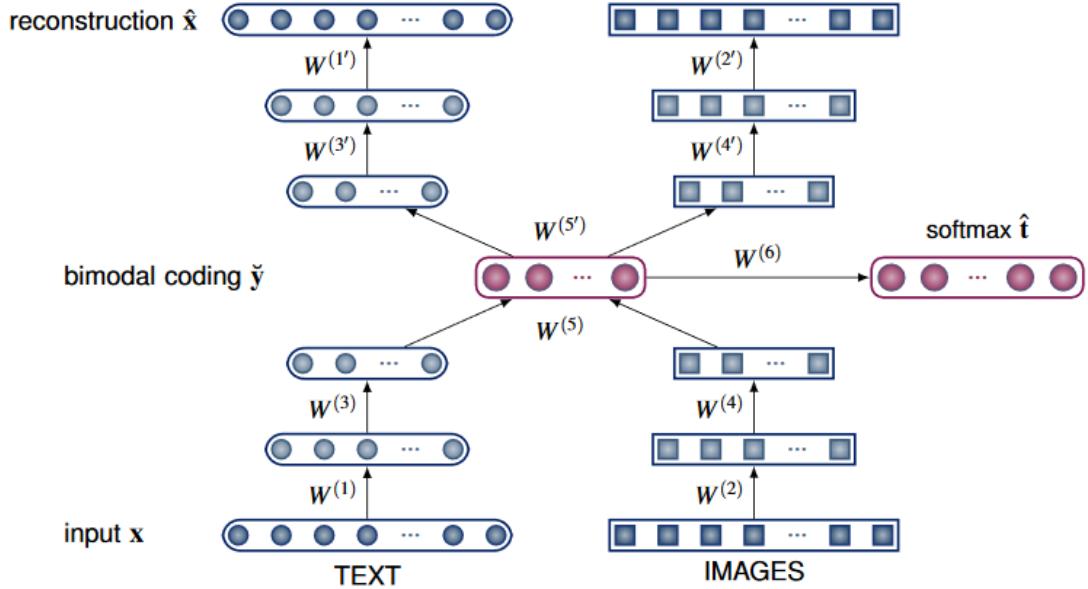


FIGURE 2.24: The stacked-autoencoder architecture of Silberer and Lapata [2014]

the autoencoder architecture to learn dependencies between modalities so that they can take advantage of learning a multimodal representation of the data conjointly instead of training two side-by-side unimodal architectures. The network was then used to capture word-pair similarity using semantic and/or visual similarity value. We can clearly see in this results intuitive phenomena: the similarity between a *cherry* and a *pineapple*

| Word Pairs              | Semantic | Visual |
|-------------------------|----------|--------|
| <i>football–pillow</i>  | 1.0      | 1.2    |
| <i>dagger–pencil</i>    | 1.0      | 2.2    |
| <i>motorcycle–wheel</i> | 2.4      | 1.8    |
| <i>orange–pumpkin</i>   | 2.5      | 3.0    |
| <i>cherry–pineapple</i> | 3.6      | 1.2    |
| <i>pickle–zucchini</i>  | 3.6      | 4.0    |
| <i>canary–owl</i>       | 4.0      | 2.4    |
| <i>jeans–sweater</i>    | 4.5      | 2.2    |
| <i>pan–pot</i>          | 4.7      | 4.0    |
| <i>hornet–wasp</i>      | 4.8      | 4.8    |
| <i>airplane–jet</i>     | 5.0      | 5.0    |

FIGURE 2.25: Word-Pair similarity performance of the stacked autoencoder architecture [Silberer and Lapata, 2014]

is highly semantic: they do not look the same but they both are fruit. On the other side, *orange* and *pumpkin* have more similarity visually, with regards to their colour, than they have semantically, despite still being closer to one another than most other words. This shows demonstrates the potential of combining visual and semantic data when performing similarity learning: what one modality might not grasp, the other will compensate.

#### 2.4.3.2 Improving invariant embedding with the use of multi-task learning

Wang and Yao [2019] explain how training a model on multiple related tasks could help the model to train better by “*exploiting both task-generic and task-specific*” information. Indeed, for any related tasks exploiting the same dataset or similar data, low-end features used by the neural network to make higher-end decisions will be shared amongst these tasks. The example presented by the authors of this paper is the classification of animals in pictures. However, the first task  $T_1$  predicts the race of a dog, the second task  $T_2$  predicts the species of some animals and third task  $T_3$  predicts the species of other animals, from a different dataset than for the other two tasks. The tasks here are always the same, but segmented within different semantic levels: while  $T_2$  and  $T_3$  are more general classification of animal species, the main task, task 1, focuses on a prediction that is harder than for the other two tasks. Despite this, they all perform very similarity computation by aggregating low-end animal features that can then help to discriminate

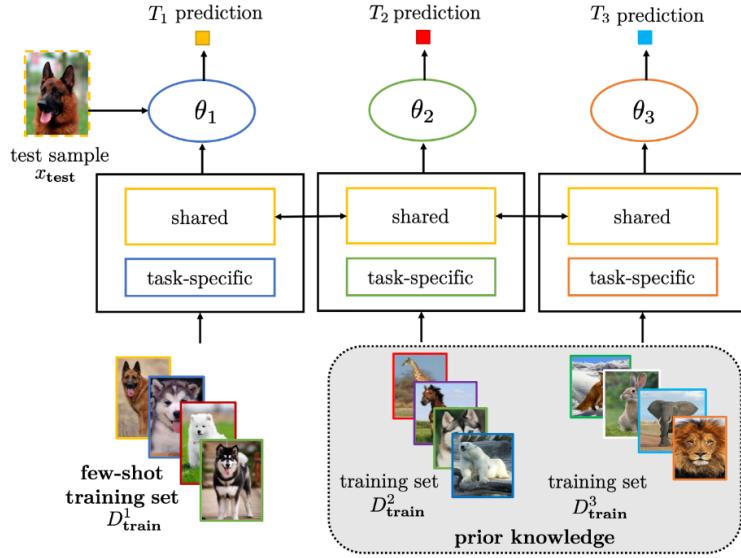


FIGURE 2.26: Multitask Illustration with parameter sharing (source: [Wang and Yao, 2019])

the correct class of the input image.

This process has, here, two main benefits:

- First, it allows the network to have way better performance: indeed, the issue with the dataset of the task  $D_{train}^1$  is that it is relatively low in samples compared to  $D_{train}^2$  and  $D_{train}^3$ . Providing additional datasets for a similar but yet different task will help the network to learn crucial shared low-end features. However, each of these tasks are different and they, consequently, hold their own task-specific weights ( $\theta_1, \theta_2$  and  $\theta_3$ ). This helps them to make use of these shared low-end features and to combine them in high-end features that are specific to their task.
- Second, it dramatically improves model generability to new unseen classes. While classification tasks will have to be generalizable to new context only, as they have a predefined set of classes, Similarity Learning algorithm does not have this concept of classes and then need to be also generalising on potentially unseen new classes. Indeed, as stated in Figure 2.28, this main difference requires similarity learning tasks to learn different types of embedding. They need a more abstract description of objects, less semantic in terms of what the object “*is*” and more how the object looks. While classification tasks will learn to detect features and match them to their corresponding class, similarity learning will need to generate instance discrimination features.

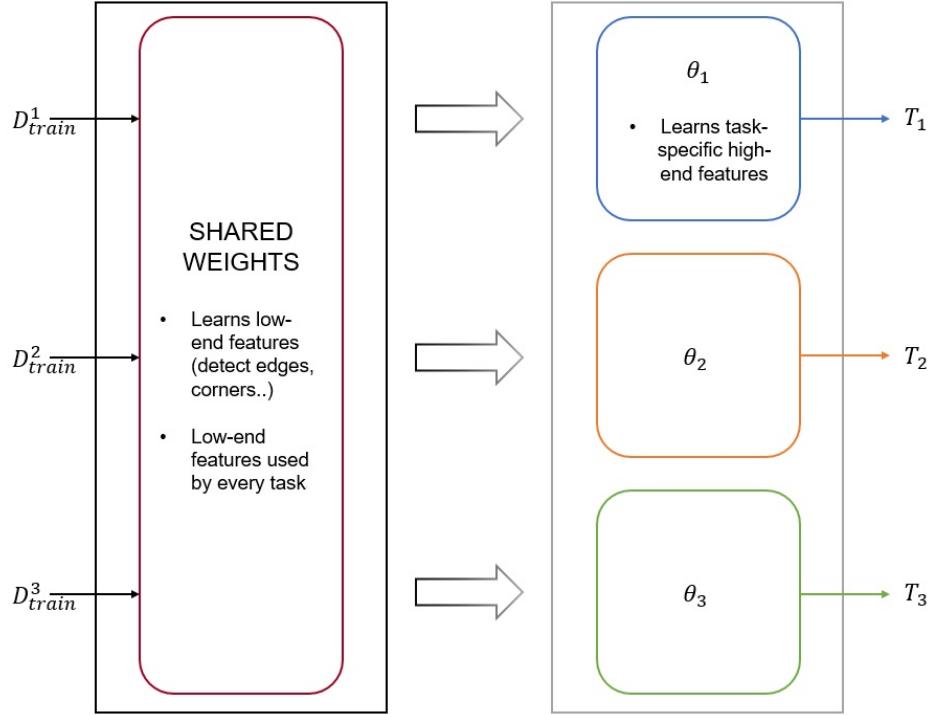


FIGURE 2.27: Illustration of weights' role in multitask learning

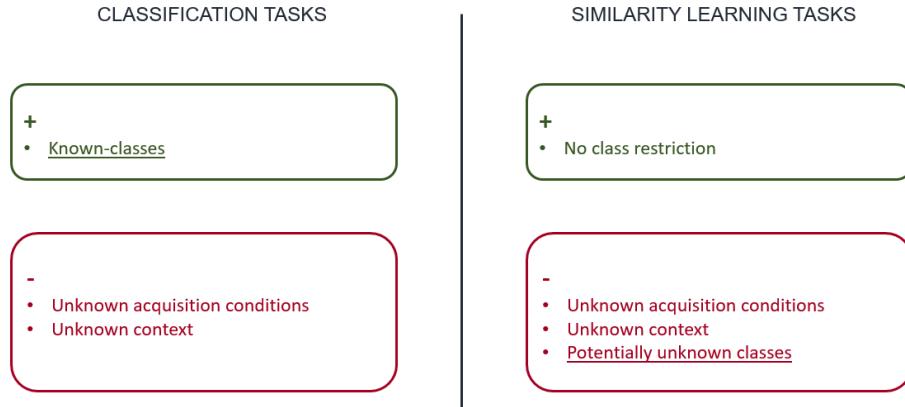


FIGURE 2.28: Comparison of some negative and positive aspects of Classification tasks and Similarity Learning tasks

## 2.5 Conclusion

Throughout this chapter, we have analysed the project under different points of view:

- A *data* point of view:

We detailed how a computer sees an image (cf. 2.1), what is Computer Vision and how its core concepts will prove themselves useful for this project. We, then,

moved on to discussing how a computer sees text and how the field of Text Mining transforms text into numbers (cf. 2.2).

- A *machine learning* point of view:

We presented the main ideas behind Machine Learning (cf. 2.3), why it is popular (and high-performing) across so many different domains by detailing the basic notions such as defining a *dataset*. We also introduced notions of Deep Learning to understand different model architectures (the Multilayer Perceptron, the CNN and the LSTM).

- A *Similarity Learning* point of view:

Then, we finally introduced the core notions of similarity learning (cf. 2.4), by presenting concrete implementations from which ours is inspired. We also detailed the multiple parameters that one may tune to improve the performance of a Deep Similarity Learning algorithm such as model architecture, training scheme or even data sampling. We also have seen how autoencoders could play an important role in learning latent embedding used to compute similarity. Especially, the multimodal approach of [Silberer and Lapata \[2014\]](#) provides good insights on how to integrate text and image data to the Siamese Neural Networks detailed above.

As seen earlier, Siamese neural networks seem to stand out in the task of Similarity learning. However, they do not represent a new architecture: they are taking advantage of the embedding capabilities that already had Deep Learning models. They are excellent at learning these invariant latent representations and it makes them very good for reducing input data to a vector of features that describes its concept, with regards to the task.

Hence, it makes the training of Similarity Network model not so hard: for most similarity tasks, pre-trained models could even be used out-of-the-shelf. The difficulty is moved from training the network to correctly sampling data to make the task difficult enough but not too hard. The data sampling strategy that one would adopt will then directly influence the performance measures as they are conditioned by the way data is sampled, making reproducibility even harder.

Furthermore, Multimodal Similarity learning is not a very common problem and we will try to dive in this concept by exploring the qualities of the embedding of each modality, the impact they have on each other once merged: will they inhibit one another or will

they have a tendency to collaborate? Is it better to train them both together from scratch or does it make sense to pretrain them on unimodal similarity learning tasks?

# Chapter 3

# Problem Context, Analysis & Requirements

In this chapter, we will go over the details of the problem initially raised by E.Fundamentals. We will explore the industrial context of this company and its influence on this project. Finally, we will discuss what was expected from this project on their part.

## 3.1 Problem context: E.Fundamentals & The DataLab

This project was the result of an industrial placement offered by the DataLab<sup>1</sup> with which I have had a scholarship. This collaboration between these 3 entities, Heriot-Watt University, The DataLab and E.Fundamentals was there to make sure to find the perfect balance between the research and industrial aspect of this project.

### 3.1.1 The DataLab presentation

The DataLab is a Scottish organisation involved in making Scotland one of the world's spearhead in data innovation. As a part of this ambition, they provide scholarship to students of 12 different universities and they organize, throughout the year, different events involving industry and facilitating collaboration between Universities, companies

---

<sup>1</sup><https://www.thedatalab.com/>



FIGURE 3.1: The DataLab logo

and students. The industrial placement from which this project originates is a demonstration of their ambitions.

### 3.1.2 E.Fundamentals presentation



FIGURE 3.2: The E.Fundamentals logo

E.Fundamentals is a company that provides to businesses an eCommerce analytics platform to help brands grow their sales by providing insights on how their products are positioned in comparison to their competitors on various retailers' website.

For that intent, E.Fundamentals keeps a database of products from different retailers by retrieving and saving how the product is presented on the website (e.g. its picture, its title, its description..). Hence, the big issue of E.Fundamentals is to be able to merge these products from different retailers that may have entered slightly different product titles or slightly different product images that would make a merge on equality impossible for most cases. For the moment, a “*static*” (i.e. not learnt) algorithm compares different product using hand-coded pattern matching rules that output a certainty score for two matching products:

- Above a given confidence measure, both products are merged.
- Below, whether both objects are detected as distinct, or the system is not sure enough and ask for human intervention that will decide to merge or not to merge different products.

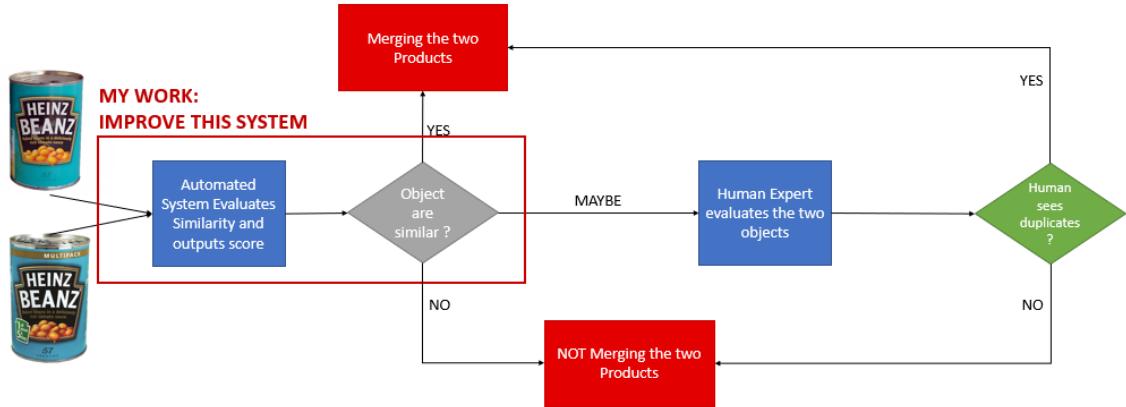


FIGURE 3.3: Original Merging System Design

Hence, to improve the original merging system (c.f. Figure 3.3), one needs to focus on improving the existing similarity function that outputs a confidence score on how similar two product data records are.

## 3.2 Problem analysis

This problem is shaped as a multimodal similarity problem: each data record is a combination of an image and a product title. Both could be used independently to compare the similarity of the two products but they could also be used conjointly. As is, the project is more of the form of a “Proof of Concept”, showing the potential of different Machine Learning methods when it comes down to similarity learning. This project should guide E.Fundamentals into the inclusion of more automated learning algorithms into their workflow. It also provides them with insights on the potential that their product database has and how much benefit they can earn from it by the use of more Data Science.

## 3.3 Project Requirements

In this section, we present the project’s goals (stakeholders, Moscow requirements..) as well as potential use cases.

### 3.3.1 Project's Goals

To define the project goals, we will use the *MoSCoW* Requirements analysis scheme as well as define the stakeholders.

#### 3.3.1.1 Stakeholders

As an industrial project in collaboration with the E.Fundamentals company, the main stakeholders come from this company:

- **E.Fundamentals personnel managing my work:** they will get **positively** affected by the project as it will provide them with better insights on ways to improve their product merging pipeline, with the addition of a statistically learnt model. If no model is found to be performing well enough for integration, at least research in the area, alongside a detailed report, will help them to evolve their product similarity detection model.
- **Database personnel,** currently merging by hand the similar products: they should get **positively** affected by the project as it should lower the amount of product they review and so free for other tasks. However, as it is not known how much time this work of merging products takes in their weekly schedule, it could have a **negative** impact on their status in the company if they are less solicited.
- **Retailers:** if this merging tool turns out to not work as well as expected and merges different product, retailers may obtain erroneous feedback on their products' performance, which would have a **negative** impact on their business as well as their decision making.

#### 3.3.1.2 MoSCoW Requirements

Acting as a prioritisation technique, the *MoSCoW* method presents 4 categories of requirements with different levels of priority: *Must*, *Should*, *Could*, *Won't*.

#### Functional Requirements

| Requirements ID | Description   | MoSCoW | Priority |
|-----------------|---|--------|----------|
| M-FR1           | This project <b>must</b> deliver a baseline system to evaluate performance of learnt Similarity Learning, no matter whether these performances are good or bad.   | M      | H        |
| M-FR2           | This project <b>should</b> deliver a functional algorithm to perform Similarity detection for the product database. This algorithm <b>should</b> come along with a well-detailed documentation relating what has been done and the functioning of the code, keeping in mind the idea that someone else will probably take over this work in the future. | S      | H        |
| M-FR3           | This project <b>could</b> make use of a new working architecture that has not yet been explored in papers, trying handcrafted meta-algorithms and innovative solutions.   | C      | M        |
| M-FR4           | This project <b>won't</b> deliver a fully integrated working application for the similarity detection. It aims at exploring data-driven approaches and having a functional algorithm at the end. The integration of the algorithm to the company pipeline will be left as future work.  | M      | L        |

TABLE 3.1: Functional Requirements Analysis

### Non-Functional Requirements

| Requirements ID | Description  | MoSCoW | Priority |
|-----------------|--|--------|----------|
| M-NFR1          | This project <b>must</b> develop an analysis of the application of Similarity Learning strategy to solve the problem of duplicate product. This analysis needs to have details on everything that worked and did not work for this context with as much detail as possible.                            | S      | H        |
| M-NFR2          | This project <b>should</b> use technologies compatible with the company's working environment (e.g. Google Cloud)  | S      | M        |
| M-NFR3          | This project <b>could</b> use an architecture which executes inference fast, not to delay the company's working pipeline in the eventuality of an integration  | C      | M        |
| M-NFR4          | This project <b>won't</b> collect external data to improve generalisation of the algorithm. All data should come from E.Fundamentals.  | W      | L        |
| M-NFR5          | This project <b>should</b> use well-known and documented Python Deep Learning libraries such as Tensorflow or PyTorch. Alongside of those, popular data science libraries such as Scikit-Learn, Numpy or Pandas <b>should</b> be considered, with existing code to be rewritten as little as possible. | S      | M        |
| M-NFR6          | This project <b>must</b> use rigorous evaluation methods that are documented and provided with the code, to make the result reproducible. These methods <b>must</b> take into account the context of the problem and need to be in phase with the needs of the E.Fundamentals staff                    | M      | H        |

TABLE 3.2: Non-Functional Requirements Analysis

### 3.3.2 Use Cases Study

**E.Fundamentals: Merges two similar products detected by the algorithm**

**Use Case ID:** 1

**Description:** The algorithm considers two products to be similar and sends this information to the E.Fundamentals code that merges them automatically.

**Primary Actors:** The E.Fundamentals code.

**Secondary Actors:** The similarity learning algorithm.

**Preconditions:** The algorithm is set up in the work pipeline of E.Fundamentals.

**Main Flow:**

1. The E.Fundamentals application sends to the algorithm two sets of image and text to be analysed.
2. The algorithm processes the two sets of image and text and outputs a confidence score that both are similar.
3. The algorithm transmits this score to the E.Fundamentals rest of the pipeline.
4. The E.Fundamentals code merges the two products.

**Post-conditions:** The two products are merged in the database.

**E.Fundamentals: two similar products detected by the algorithm, but not with enough confidence**

**Use Case ID:** 2

**Description:** The algorithm considers two product to be similar, but not with enough confidence, and sends this information to the E.Fundamentals code that asks for a human operator to make the decision.

**Primary Actors:** The E.Fundamentals code.

**Secondary Actors:** The similarity learning algorithm, E.Fundamentals database engineers.

**Preconditions:** The algorithm is set up in the work pipeline of E.Fundamentals.

**Main Flow:**

1. The E.Fundamentals application sends to the algorithm two sets of image and text to be analysed.
2. The algorithm processes the two sets of image and text and outputs a confidence score that both are similar but that the algorithm is not fully sure.
3. The algorithm transmits this score to the E.Fundamentals rest of the pipeline.
4. The E.Fundamentals code transmits the two products information to a database engineer.
5. A database engineer reviews the two products and merges them.

**Post-conditions:** The two products are merged in the database.

### 3.3.3 Evaluation

As a “*Proof of Concept*”, our project needs to present results in an interpretable way that could easily illustrate how it performs. As a wide range of algorithms and settings were explored, we need them to be comparable in a way that is meaningful both on the research point of view as well as on E.Fundamentals’ point of view.

#### 3.3.3.1 Evaluation discussion

In [2.3.4](#), we saw the important role played by performance metrics in evaluating a system. Our automated Similarity Learning system will potentially perform two different kinds of failures during the matching process: False Positives (i.e two different products considered similar) or False Negatives (i.e two similar products considered different):

- **False Negatives** would not have a very bad impact on the overall system (cf. [Figure 3.3](#)). Indeed, as long as two objects are similar and the algorithm just has a doubt about it, then a human operator will be able to take over and merge if need be. FNs would be a difficulty if our system is certain that two objects are different while they, truly, are similar. In this situation, we would end up with two similar objects being left untouched in the database, without even having a human operator verifying this prediction. Nevertheless, this kind of problem is not as impactful as false positives;
- **False Positives** would, indeed, have an impact on the system: merging two products that are supposed to be different would bias the analytic lead by the E.Fundamentals team (for example, comparing the prices of a product and a pack of the same product would be very noticeable and could lead to misleading conclusions from the E.Fundamentals tools).

#### 3.3.3.2 Retained metrics: F-beta score & n-way testing

**F-beta score** A useful metric would be one penalising in priority the *False Positives*. At first sight, the precision measure seems like an ideal metric as it can efficiently report False Positives. It would tell, from all items classified as being the same products, how many of them are correctly classified as being the same products. Despite this, it would

be unable to report anything about FN: missed similar products. For that intent, we would need to introduce as well the Recall measure. The F1-score seems like a good compromise and is defined as follows:

Given that we defined the precision and recall as  $precision = \frac{TP}{TP + FP}$  and  $recall = \frac{TP}{TP + FN}$ ,

$$f1 = 2 * \frac{precision \times recall}{precision + recall}$$

However, we see the main weakness of the F1-score: it weights the recall and the precision in the same way. A better alternative, that would prioritise precision without disregarding recall is the F-beta score:

$$f_\beta = \frac{(1 + \beta^2) \times (precision \times recall)}{\beta^2 \times precision + recall}$$

The addition of the  $\beta$  parameter weights the recall: the lower  $\beta$  is, the less important the recall is<sup>2</sup>.

In addition to the F-beta score, another good metric would be one that simulates the final function of the algorithm: finding similarity between one object and a list of candidates. To have a test environment similar to this usage environment, the n-way testing seemed to be an ideal candidate.

**N-way testing** N-Way testing is a method where an image is compared to group of N other images, amongst which only one is matching. We will call the image compared to the N others the *anchor image*. Given a dataset  $D$ , split into  $D_{train}$  and  $D_{test}$ , the sampling for the anchor image and for the N other images has to be made within the boundaries of the dataset split: in other words, if the algorithm is tested on  $D_{test}$ , both the anchor image and the N images will be sampled from  $D_{test}$ . The N-Way testing algorithm is as follows:

Common values of N are found between 10 and 20. The lower the N, the bigger the n-way test value should be. Hence a good balance needs to be found between having a big enough value of N to keep the dataset distribution and small enough to keep computation time reasonable.

---

<sup>2</sup>note that if  $\beta = 1$ , we obtain the formula of the F1-score

---

**Algorithm 2** N-Way Testing algorithm

---

**Input:***D*: Evaluation Dataset*N*: Number of samples to use at testing time*M*: Embedding extraction model

```

1: function N WAY TEST(D, N, M)     $\triangleright$  The generic n-way testing pseudo-algorithm
2:   csuccess = 0
3:   ctotal = 0
4:   for each dataset item d in D do            $\triangleright$  Looping over all test examples
5:     s  $\leftarrow$  similar(d, D)                   $\triangleright$  Sample one similar item
6:     dis  $\leftarrow$  dissimilar(d, D, N - 1)       $\triangleright$  Sample N-1 dissimilar items
7:      $\delta_s = \|M(d) - M(s)\|_2$ 
8:      $l_{\delta_{dis}} = \{\|M(dis_0) - M(s)\|_2, \|M(dis_1) - M(s)\|_2, \dots, \|M(dis_{N-2}) - M(s)\|_2\}$ 
9:     if  $\delta_s < \min(l_{\delta_{dis}})$  then
10:       csuccess = csuccess + 1
11:     end if
12:     ctotal = ctotal + 1
13:   end for
14:   return csuccess/ctotal     $\triangleright$  Return the proportion of successes in N-Way testing
15: end function

```

---

# **Chapter 4**

# **Professional, Legal and Ethical Issues**

In this chapter, we will discuss the eventual Professional, Legal, Ethical and Social issues encountered in this project. While limited, this project is linked to a business whose job is to report on brand performance in retail stores by retrieving the prices of their product across multiple retailer online stores.

## **4.1 Professional**

Considering the industrial context of this research, professional standards with regards to the deliverables were expected. As a computer science project, I made sure that this work follows the Code of Ethics and Professional Conduct of the *Association for Computing Machinery* as well as the four key principles of the Code of Conduct of the *British Computer Society*:

- To Make IT for everyone;
- To show what I know, and learn what I do not;
- To respect the organisation (i.e E.Fundamentals) and the individuals (i.e E.Fundamentals personnel as well as my academic advisor, Dr IRELAND) I work for;

- to keep IT real, professional and to pass IT on.

By these two codes of conduct, I also made sure that licence of any third-party libraries, software or product has been strictly followed. No product of any kind was used in conditions that are not permitted by their licence. Intellectual property has been respected so that any information and citation will be sourced.

## 4.2 Legal

As presented in the Professional section, no product will be used in an illegal way that would violate their licence.

Concerning the E.Fundamentals dataset, their dataset is not public and should then not be leaked. In the same time, it consists of the list of product description retrieved directly from retailers' websites: no private information is embedded in these datasets.

## 4.3 Ethical

No contact with the real user or user-related data is made within this project. The only ethical issues that may arise from this project come from the industrial context of E.Fundamentals as well as the ethical issues that some of its clients may be facing. However, this is not a concern for this project as no direct contact with the clients are made and this project will be only to improve E.Fundamentals database engineers' working conditions.

## 4.4 Social Issues

While direct social issues do not exist within this project, with this exact application (i.e. product similarity identification), multiple scenarios involving the use of a multimodal algorithm, in the same manner, can be imagined, especially in sensitive environments such as the medical/pharmaceutical domain:

Given a pharmacist mistakenly taking a wrong product off his shelf, a similarity learning algorithm could compare the product chosen from the one asked in a prescription. In

this situation, direct social benefits, and issues, may arise from the use of the algorithm. So even if the working context is without any social issues, the idea of a multimodal similarity learning algorithm can have noticeable social impact.

# Chapter 5

## Implementation: Dataset & Models

In this chapter, we will describe the data, the cleaning process it went through and the preprocessing pipeline for each modality as well as what a data sample looks like. We will also present the deep architectures built for different experiments.

### 5.1 Data

#### 5.1.1 Data retrieval

The data from this project comes from a Google Big Query database of E.Fundamentals. To retrieve it and process it, a query was written, extracting relevant products from the last 5 years. Multiple criteria were imposed:

- Product information had to be in English;
- Product had to be matched with another product considered similar;
- No duplicate product was retrieved. For duplicate products, a “*keep first*” strategy was followed.

Product information was then saved into a .csv file. Each row of this .csv file corresponds to a first product that was called the “*true product*” while the second, matched to the

first was called the “*extracted product*”. Information from both products did not come from the same DB table: true products are products that were reviewed by the brand that commercialise them so that they fix their image and name. Indeed, the company has found that a lot of retailers tend to take some liberties with regards to products’ appearance on their website and they do not always have the name and image provided by the manufacturer.

The extracted products are in another table where the products have not been “*cleaned*” by the manufacturer and may consist of a changed title and images.

Multiple product descriptors were retrieved:

- True product title;
- True product image (under the form of a *URL* pointing to a .res image stored in Google Cloud);
- True product quantity;
- True product brand;
- True product retailer
- Extracted product title;
- Extracted product image (array of links to every image displayed for the product in the retailers’ website, also stored in Google Cloud);
- Extracted product quantity;
- Extracted product retailer;

This resulted in around 6500 rows of data, meaning 6500 *true products* and 6500 *extracted products*.

### 5.1.2 Data cleaning

However, as is, a few data incoherence existed, such as:

- Wrongly matched products: some products were considered as a match in the database but were not;
- Some extracted products did not have images or their images were corrupted.

**Remove products with corrupted image** For that intent, I looped over all the dataset and removed every row where the extracted image would throw an Exception when accessed, or if it was just missing. This cleaned the dataset of around 2000 rows.

**Remove wrongly matched product** At first, I tried finding every wrongly matched product on an iterative process with the hope of finding pattern (i.e. a single item was always matched to random other items) but without success.

I finally decided to exploit the product titles: once processed (cf 5.2.1.1), if the *true product name* and the *extracted product name* were not having at least 2 words in common, they were discarded. A risk existed as two actual similar products could have different names but the risk of keeping them was bigger, so I removed these rows: 165 out of the 4300 rows left.

At the end of these two processes, I had left a bit more than 4000 rows of products, which is not much but was almost noise-free in term of content. Hence, this relative absence of noise was a good sign for Data Augmentation that would prove itself to be highly efficient.

### 5.1.3 Dataset characteristics

After every processing, the final characteristics of the dataset are:

- 4119 rows for 11 columns;
- 76 different brands;
- A 2.4Mo .csv file;
- Around 499MB worth of images.

As we can see, this is not a huge dataset. It does not have a lot of elements. however, as every product is unique, extensive Data Augmentation is possible without creating too much overfitting and without minimising the generalisation properties of the network.

An extract of his content can be seen in Figure 5.1, ProductID were censored, for security issue.

| Product_ID | True_Product_N  | True_Product_Url     | True_Product_Q | True_Product_B  | Product_Retailer | Extracted_Prod   | Extracted_ProdUrl  | Extracted_Net_C | Extracted_Net_C | Extracted_Retailer |
|------------|-----------------|----------------------|----------------|-----------------|------------------|------------------|--------------------|-----------------|-----------------|--------------------|
| XXX        | galaxy smooth n | https://storage.g... | 360g           | galaxy          | Tesco            | galaxy milk choc | https://storage... | 0.3600          | KG              | Tesco              |
| XXX        | james wellbelov | https://storage.g... | 225g           | james wellbelov | Viovet           | james wellbelov  | https://storage... | 0.2250          | KG              | Viovet             |

FIGURE 5.1: Extract of the dataset

Hence, the column exploited by the deep learning models are the product titles and images, providing data records of this aspect: Notice in Figure 5.2 the difference in the

Product 1:



starburst original fruit chews candy bag forty-one ounce

Product 2:



starburst fruit chews original party size

FIGURE 5.2: Example of a data sample: true product on the left, extracted product on the right

image between the true and the extracted product: the true product seems to have a cleaner more modern image while the extracted product seems to have an older version. At the same time, the title of the product is way more descriptive in the true product than on the left, adding, in particular, the quantity, here “41 ounce”.

## 5.2 Models

Variations in the models emerge from the fact that we are dealing with multimodal data: to explore each modality and their mutual interactions, multiple models have been developed, each to try to capture different data similarity profiles. However, despite their differences in implementation, they all follow the same meta-structure: Hence, changes

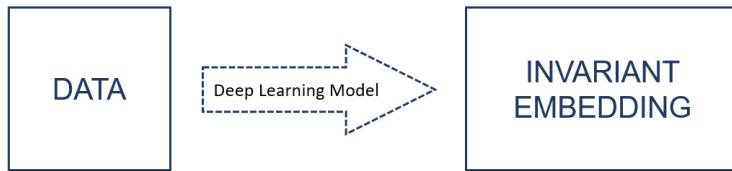


FIGURE 5.3: Abstract representation of the embedding extraction process

in the data modalities will impact the deep learning architecture but not its function. Every deep learning architecture will here serve the same purpose of describing the input

data with invariant embedding, that would help to recognize similarity within data taken under different conditions.

As stated before, multiple models have been developed to explore the parameter space of our problem. In particular, we have explored the following models:

- A *unimodal Siamese Recurrent Neural Network*, to produce a similarity score based on the product title;
- A *unimodal Siamese Convolutional Neural Network*, to produce a similarity score based on the product image;
- A *multimodal Siamese Neural Network*, to produce a similarity score using a combination of the product title and the product image.

The retained architecture and parameters of each of these networks were retained after empirical experimentation. The main objective of their development was to compare the performance of each modality and see how well they perform when learnt conjointly. We will now go through each of these models and detail the retained architecture

### 5.2.1 Unimodal models

#### 5.2.1.1 Siamese RNN

The full text processing pipeline detailed in Figure 5.4 presents the steps to go from “*Product Title*” to the vector embedding used to compute distances between product titles.

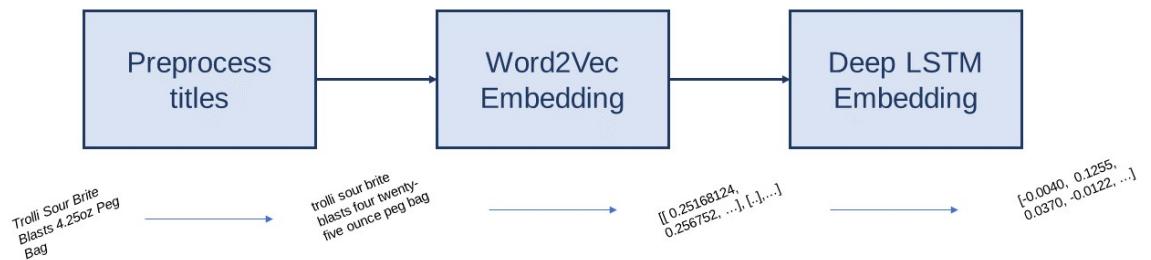


FIGURE 5.4: Text Processing Pipeline

**Preprocessing** When preprocessing text, we apply 5 basic transformations:

1. The words are set to lower-case letters;
2. Numbers are converted into their word equivalent (e.g. “1” becomes “one”);
3. Non-alphanumeric characters are removed (e.g. “M&Ms” is transformed into “MMs”);
4. English stop-words are removed (e.g. “and”, “the” ...);
5. Using a regular expression formula, we split any number attached to a letter. In the case of product titles, it usually indicates quantities. The unit of this quantity is then extracted and transformed into its full word version. (e.g. “120g” becomes “one hundred twenty grams”).

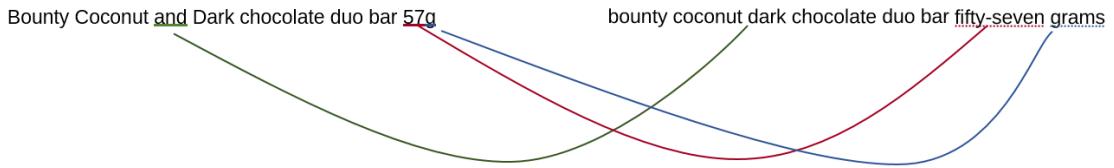


FIGURE 5.5: Text Preprocessing Example

A concrete example of this preprocessing applied to text is displayed in Figure 5.5. This processed sentence is then transformed into a list of vector representation for each of its word using a Word2Vec model.

The gensim implementation of Word2Vec was used, with an embedding size of 20: every word will then be transformed into a vector of size 20.

The Word2Vec model is then trained for 30 epochs (30 complete loops over the dataset of product titles). As is, our word corpus represents around 8000 words. This Word2Vec model would need to be retrained as new products are added.

**Siamese LSTM Architecture** For the Siamese Deep Learning architecture, it is designed as a stack of 3 Bidirectional-LSTMs as shown in Figure 5.6. It takes, as an input, an array of vectorised words of size  $n * 20$  where  $n$  is the number of words in the Product Title and 20 the vector size of each word, once transformed with Word2Vec. The model then outputs a fixed-length sequence of size 16. This vector is then used to

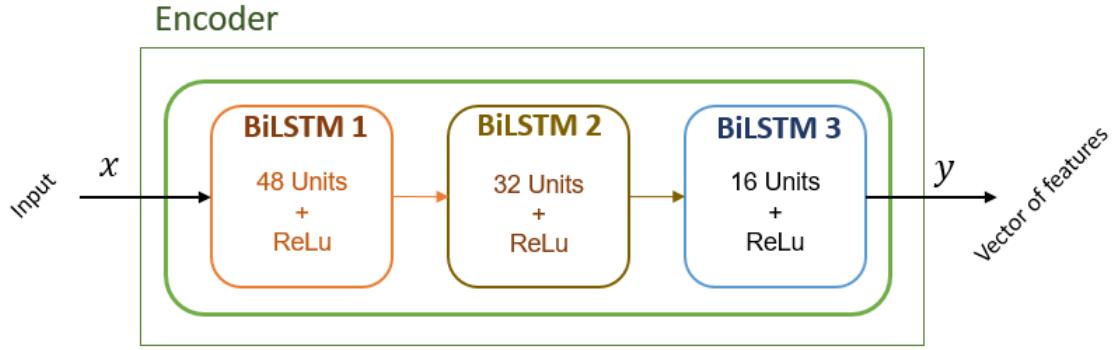


FIGURE 5.6: BiLSTM architecture used

compute distances between input sentences. This architecture is cloned once or twice, depending on the training loss chosen.

### 5.2.1.2 Siamese CNN

For the siamese convolutional neural network, a pretrained VGG16 [Simonyan and Zisserman, 2014] architecture has been chosen. The use of transfer learning helps the algorithm to converge faster and makes learning more stable. The architecture of the

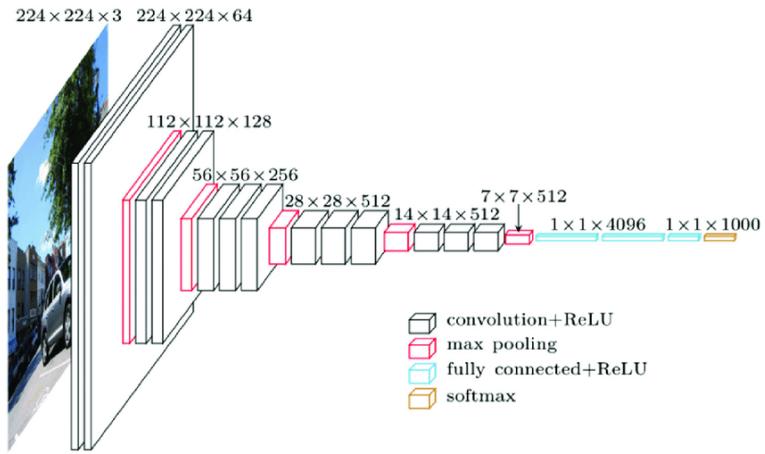


FIGURE 5.7: Original VGG16 Architecture (source: [Nash et al., 2018])

network initially takes a 224-by-244 input image (cf. Figure 5.7) but it has been modified to 256-by-256 for this work. Furthermore, from all the fully-connected layers of the initial CNN, the “*head layers*” (i.e. the fully connected layers) have been removed. Indeed, only the convolutional and pooling layers were kept as they have the role of feature extraction that will create our embedding vector. Then, new fully-connected

---

```
(0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(1): ReLU(inplace=True)
(2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(3): ReLU(inplace=True)
(4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
(5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(6): ReLU(inplace=True)
(7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(8): ReLU(inplace=True)
(9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
(10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(11): ReLU(inplace=True)
(12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(13): ReLU(inplace=True)
(14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(15): ReLU(inplace=True)
(16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
(17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(18): ReLU(inplace=True)
(19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(20): ReLU(inplace=True)
(21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(22): ReLU(inplace=True)
(23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
(24): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(25): ReLU(inplace=True)
(26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(27): ReLU(inplace=True)
(28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(29): ReLU(inplace=True)
(30): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
```

---

LISTING 5.1: VGG16 Architecture with layerwise details

layers were added to “compress” the output of the VGG16 architecture into a 256-sized embedding vector.

The stack of convolutional, pooling and activation layers used are displayed in Listing 5.1.

### 5.2.2 Siamese Multimodal neural network

The siamese multimodal neural network combines the two model presented before into one. It takes as input an image and a text. The preprocessing of either of those are respectively the same than for the Siamese CNN and the Siamese LSTM.

#### 5.2.2.1 Architecture

The network computes the embeddings of the input product title and product image using paralleled stacked BiLSTMs and the VGG16 architecture. Then, it concatenates their weighted mutual output and merges the embedding of each of the two modalities using a Fully-Connected layer as an output, that produces the final vector of features.

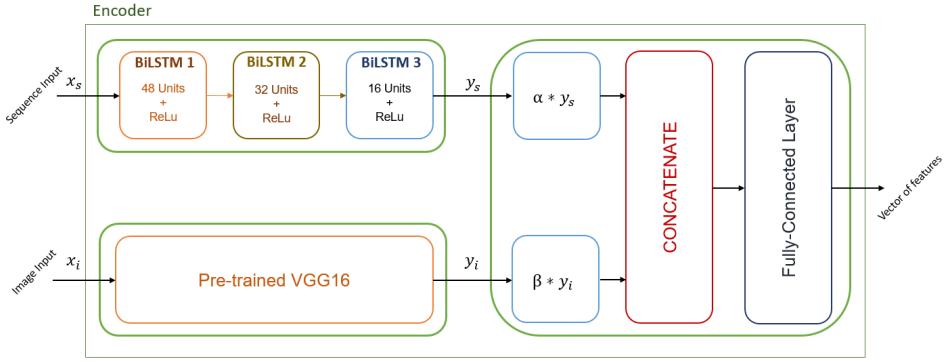


FIGURE 5.8: Multimodal Network architecture

### 5.2.2.2 Modalities combination strategy

The strategy employed to combine the different modalities were chosen through trial & errors. The retained solution uses the direct output of the embedding of the different modalities, without any fully-connected layer. These embeddings are then weighted using parameters that are gradient-dependant, meaning that they will be trained through backpropagation. Then, the weighted embedding are concatenated and are finally passed to a fully-connected layer that will compress them.

A simple concatenation of the output of each modality's embedding would suppose that they are equally as important to make a decision. However, while it is expected for each modality to contribute to the decision, it is highly unlikely that they would do so on the same scale. Hence, these weights help the network to inhibit one of the modality while, at the time, stimulate the other. Furthermore, the analysis of these weights at the end of training will help to understand the contribution of each modality to the final decision. This improves model interpretability, which is usually a drawback of Deep Learning.

As stated above, different solutions were explored:

- Simple concatenation of each modalities embedding output: it resulted in an unstable learning procedure, that could be justified by an imbalance in embeddings' dimensions. Indeed, the image embedding dimension was larger than the text embedding's, and this by multiple orders of magnitude;
- Non-weighted modalities merging: this led to the idea of weighting each modality at merging time. While not necessarily bad, it was seen as a potential area of improvement and indeed was.

### 5.3 Code architecture & Program overview

The root of the project is split into 3 folders:

- `/data`: holds every .csv file corresponding to the different versions of the dataset through time;
- `/notebooks`: holds notebooks used for experiments as well as for quick exploratory data analysis (EDA), data cleaning as well as data retrieval;
- `/scripts`: contains every Python classes essential to the application; work standalone from the notebooks, training and inference can be run from there.

#### 5.3.1 Scripts folder details

The main point of focus of my work can be found in `/scripts/models` where I store every model coded using PyTorch 1.5.0. 4 folders can be found:

- `/scripts/models/imageBasedModels`: contains every model and PyTorch compatible dataset class (`torch.utils.data.Dataset`) for unimodal similarity learning with product images;
- `/scripts/models/multimodalModels`: contains every model and PyTorch compatible dataset class (`torch.utils.data.Dataset`) for multimodal similarity learning;
- `/scripts/models/preprocessing`: contains the Word2Vec preprocessing module;
- `/scripts/models/titleBasedModels`: contains every model and PyTorch compatible dataset class (`torch.utils.data.Dataset`) for unimodal similarity learning with product titles;

The PyTorch architecture, inheriting from `torch.nn.Module` can be found in the root of this directory as well, in 3 different files: `siameseConvolutionalNetwork.py`, `siameseMultimodalnetwork.py`, `siameseRecurrentNetwork.py`.

When executing the classes found in the model folders, using `python productImageModel.py` for instance, a training session with default parameters is launched.

### 5.3.2 Demonstration website

For the purpose of E.Fundamentals demonstration, a small web-interface was edited using Flask. This interface, presented in Figure 5.9, samples randomly

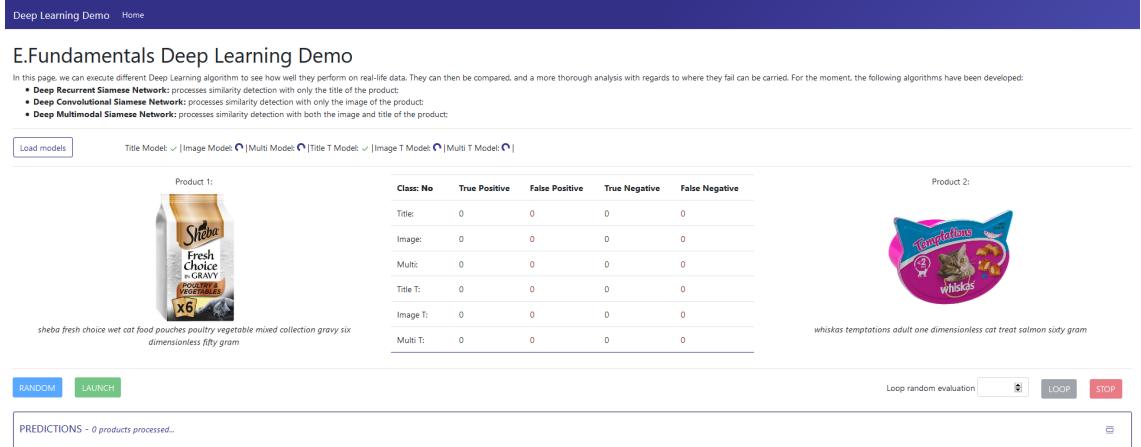


FIGURE 5.9: Screenshot of the demo website

products from the dataset (matching & non-matching) and run the prediction of 6 different deep learning algorithms (3 of them trained with contrastive loss, 3 of them with triplet loss). The results are then displayed in a confusion matrix with 6 rows, one per model. This allows the user to have quick insights on the performance of the models, for every possible metrics derived from the confusion matrix (Accuracy, Precision, Recall...).

Furthermore, a dropdown menu lists every product similarity predictions performed by the 6 models. Acting as a prediction history, it allows the user of the interface to investigate concrete examples of misclassified products.

# Chapter 6

## Experimental Design & Results

In this chapter, we will present our experimental setup, our evaluation strategies as well as analysis of these results. All the experiments are made with regards to the initial research questions.

### 6.1 Experimental Design

In this section, I will describe my experiment as well as my working environment, to make my results easier to reproduce.

#### 6.1.1 Software & Hardware

Development has been done using Windows 10 with an Anaconda distribution of Python 3.7. The table 6.1 illustrates the version of the main libraries used<sup>1</sup>.

In term of hardware, I worked on a laptop Dell G5-5590 equipped with an I7-8700, 32Go of RAM, 1To HDD + 256 SSD and an NVIDIA 2060 6Go GPU. I did not lack RAM during my work but VRAM, limited to 6Go, was easily full, making my range of experiments limited in terms of Batch Size: indeed, the bigger the batch size, the more data is loaded in memory at the same time, alongside the model. For the BiLSTM model, they are very lightweight so this was no issue but for anything involving fully-connected layers or convolutional layers, my processing capabilities were limited. Furthermore, my

---

<sup>1</sup>The libraries noted with a (\*) are Python libraries, the other are usual software/drivers

| Library          | Version     |
|------------------|-------------|
| PyTorch (*)      | 1.5.0       |
| Scikit-learn (*) | 0.23.1      |
| Numpy (*)        | 1.18.2      |
| Pandas (*)       | 1.0.3       |
| PIL (*)          | 5.4.1       |
| OpenCV2 (*)      | 4.1.1       |
| torchvision (*)  | 0.6.0+cu101 |
| cuDNN            | 7604        |
| CUDA             | V10.0.130   |

TABLE 6.1: Software version listing

computation speed was limited.

For example, is displayed in Table 6.2 a list of GPUs and the maximum batch size they support on training different State-of-the-Art (SOTA) architectures. Despite these

| Model / GPU  | 2060 | 2070 | 2080 | 1080 Ti | 2080 Ti | Titan RTX | RTX 6000 | RTX 8000 |
|--------------|------|------|------|---------|---------|-----------|----------|----------|
| NasNet Large | 4    | 8    | 8    | 8       | 8       | 32        | 32       | 64       |
| DeepLabv3    | 2    | 2    | 2    | 4       | 4       | 8         | 8        | 16       |
| Yolo v3      | 2    | 4    | 4    | 4       | 4       | 8         | 8        | 16       |
| Pix2Pix HD   | 0*   | 0*   | 0*   | 0*      | 0*      | 1         | 1        | 2        |
| StyleGAN     | 1    | 1    | 1    | 4       | 4       | 8         | 8        | 16       |
| MaskRCNN     | 1    | 2    | 2    | 2       | 2       | 8         | 8        | 16       |

\*: GPUs without enough memory to run the model

TABLE 6.2: Comparative table of GPU performance (source: [lambdalabs.com](https://lambdalabs.com))

limitations, I was still able to run most of the experiments I wanted as other hyperparameters can be tuned without having any impact on VRAM Usage.

### 6.1.2 Experimental Setup

We will now describe the experimental setup for the 3 main Deep Learning models developed: the two unimodal and the single multimodal siamese networks.

#### 6.1.2.1 Data setup

Multiple data related training procedure were used. There was:

- Dataset split: the dataset was split in 3 sets: training, with 80% of the dataset, validation, with 10% of the dataset and test with 10% of the dataset. This split process was kept for each model, preserving the same data samples in each split, for comparison purposes.

- Data augmentation: for the convolutional unimodal network training as well as the multimodal network training, the dataset was augmented by a factor of 16. The augmentation process included 3 transformations randomly applied:
  - **Color Jitter** with hue *jittered* by a factor uniformly sampled from  $[-0.05, 0.05]$  and saturation also *jittered* by a factor uniformly sampled from  $[-0.05, 0.05]$ ;
  - **Image rotation** with a rotation angle uniformly sampled at random from  $[-20, 20]$  degrees interval. The empty pixels left from the rotation processed are filled with white, usual background color of product images;
  - **Image Perspective** modification with a distortion scale of 0.5 and a probability of happening of 0.5. Empty pixels were also filled in with white.

When training the multimodal and convolutional unimodal network, the data was augmented with a factor of 16. Examples of effect of augmentation can be seen in Figure 6.1.



FIGURE 6.1: Example of data augmentation

### 6.1.2.2 Training procedure

We will now go through the training details of each of the 3 models. For every experiment, the same procedure was kept to have comparable results spanning over different experiments.

The retained loss function for similarity learning was the *Triplet Loss* for its performance. Performance comparison over multiple loss will not be reviewed in this thesis but the triplet loss has proven itself to be more efficient at generating embedding than

other losses, including the Contrastive Loss and the BCELoss with sigmoid layer output. More details of these losses were reviewed in 2.4.

There were two different types of training: one specific to the Siamese recurrent network and another one applied to the Multimodal Siamese Network and the Convolutional Siamese network:

- **Training of the Siamese Recurrent Neural Network:** the training of the BiLSTM Siamese network used an Adam optimizer [Kingma and Ba, 2014] with a learning rate of  $1e-3$ ,  $\epsilon$  of  $1e-8$  for numerical stability,  $\beta$  values of 0.9 and 0.999. Neither weight decay nor AMSgrad was used. The learning rate was reduced using a `ReduceLROnPlateau()` that reduces the learning rate by a factor of 0.1 whenever from one epoch to the next, validation loss has not decreased. Additionally, a batch size of 16 was used. Although a bigger batch size was possible to be used (up to 256), experiments showed that a much smaller batch size induced better performance.
- **Training of Siamese CNN & Multimodal CNN:** the training of these two networks had a lot in common in terms of hyperparameters. They both used a Stochastic Gradient Descent algorithm as an optimizer with layer-wise learning rates. Feature layers (convolution & pooling layers) had their learning rate set to  $1e-5$  while top layers, generating embeddings, had their learning rate set to  $1e-1$ . This is justified by the fact that the convolutional layers are already pretrained, oppositely to the fully-connected layers (FC) on top of the network that generate the embedding. hence, we want faster training of these layers compared to the feature layers. Furthermore, learning rate decay was scheduled, for both feature and FC layers using  $f(LR) = 0.99^{epoch} * lr$ . For the batch size, hardware limitations (cf. Table 6.2) forced the use of a batch size of 8 products.

As mentioned above, the VGG16 network was weights pre-trained on ImageNet for faster convergence. Additionally, the training of the deep learning models involved the use of sampling hard examples within each mini-batch. Illustrated in Figure 6.2, this increases convergence and improves model performance (cf 2.4)<sup>2</sup>.

---

<sup>2</sup>The GitHub repository used to perform online hard batch sampling can be found [here](#)

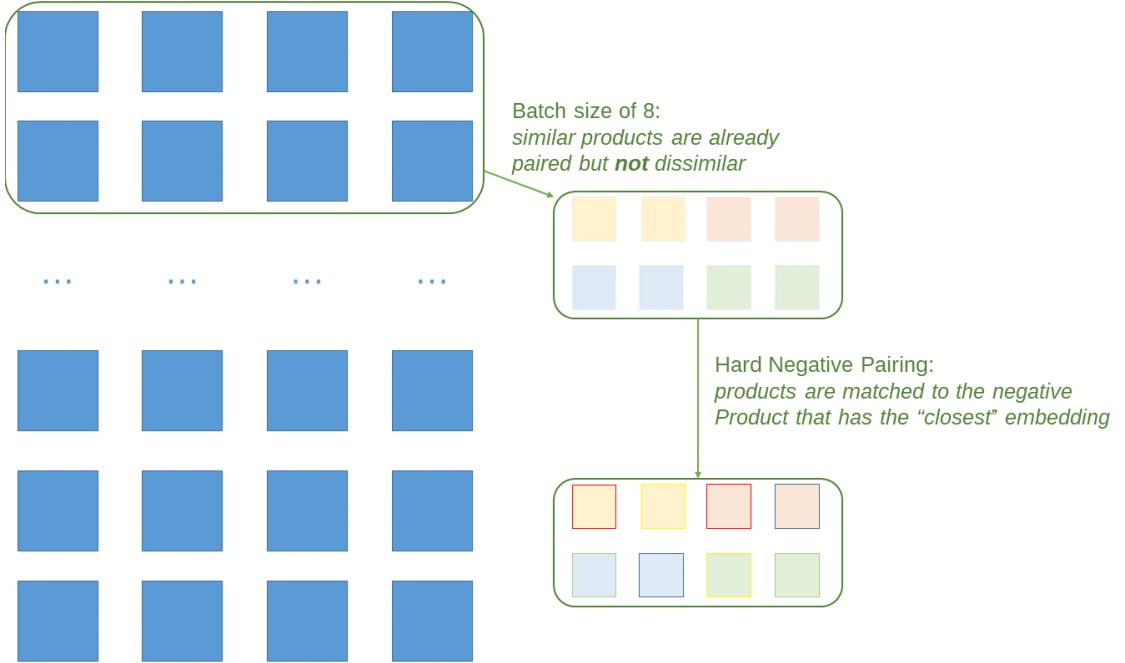


FIGURE 6.2: Online Mini-Batch hard sampling illustration

### 6.1.3 Research Questions & Hypothesis

Above we have described our evaluation strategy from the perspective of the end-user. Here we focus on the research questions that underly our proposed technique. The research questions we try to answer as well as the underlying hypotheses behind these questions.

#### 6.1.3.1 Research Questions

- (1) Does a deep learning model outperform simpler and more common machine learning algorithms with handcrafted features for product similarity learning? To what extent?
- (2) To what extent does having a siamese multimodal model give better performance than unimodal models? What is the impact on the resulting embedding ?
- (3) How much does the Convolutional Neural Network's weights from the multimodal model differ from the weights of the same model but within the unimodal architecture ?
- (4) Can multitask learning improve model performance ?

### 6.1.3.2 Hypotheses

From this research question, we can now define our hypotheses:

1. **If** a deep learning model is used to predict products similarity, **then** it will perform better, according to our evaluation metrics, than other more common and less complex machine learning algorithm as it can find a pseudo-optimal similarity measure.
2. **If** a multimodal model is used **then** it will show better performance than unimodal models as it will capture and extract features across multiple modalities at once, impacting the embedding.
3. **If** a siamese model uses multimodal data to train its image feature and text feature extractors, **then** it will find different visual and textual features by developing different weights.
4. **If** a siamese neural network is trained or pre-trained using multitask learning **then** it will have better performance and generalisation than another model trained without multitasking.

## 6.2 Results

### 6.2.1 RQ1: *Does a deep learning model outperform simpler and more common machine learning algorithms with handcrafted features for product similarity learning? To what extent?*

Considering the research question N° 1, we suggested the following hypothesis:

*“If a deep learning model is used to predict products similarity, **then** it will perform better, according to our evaluation metrics, than other more common and less complex machine learning algorithms as it can find a task-specific data embedding to efficiently compute similarity measure.”*

To verify this hypothesis, an ML-based model has been designed. We will first introduce it and then present the results of the experiments.

### 6.2.1.1 Baseline Model Presentation

As a baseline model, we used a Decision Tree Classifier over two handcrafted feature: one crafted using image features, the other using text features.

The processing of the input data is described in Figure 6.3. The two handcrafted

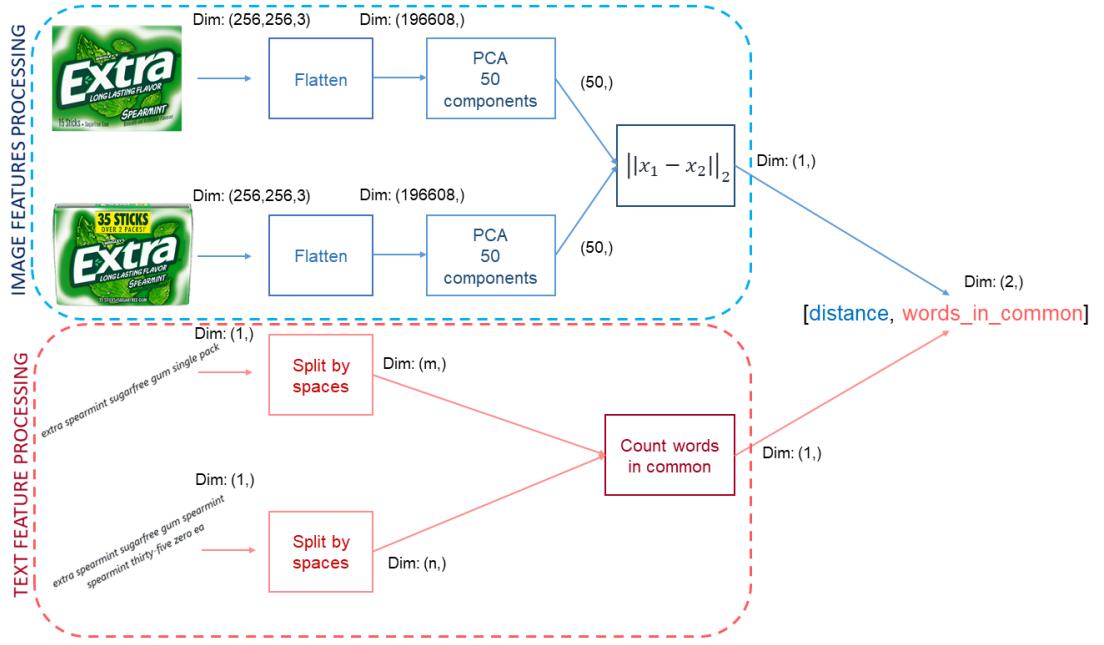


FIGURE 6.3: Handcrafted features for Decision Tree Classification

features have different characteristics and preprocessing pipelines:

- **Distance calculation of PCA-reduced images:** the first feature, extracted from the images, uses a PCA, fitted over the training set, to reduce the RGB input image (initially size 256 by 256 by 3) into a vector of 50 values. We then calculate the  $l_2$ -norm of the subtraction of the two vectors. This distance measure acts as one of the two features<sup>3</sup>;
- **Common word count:** the second feature, extracted from product titles, is a count of common words in product titles, once they are split by spaces.

From these 2 features, 3 different trees were built: two unimodal trees (one per modality), to compare with the unimodal Deep Learning models as well as one multimodal tree.

<sup>3</sup>No normalisation is done over the distance feature, values are then quite high, in a matter of thousands. If linear models were to be used, a normalisation process would be needed.

The architecture of the trees can be seen in Appendix B. Depth of the trees was limited to 10 and pruning has been performed to limit overfitting.

### 6.2.1.2 Experimental results

A total of 12 experiments were run: 2 per models (one involving the F-beta score measure, the other involving the N-way testing with N set to 16), for a total of 6 models (3 deep learning architectures, presented in 5.2 and 3 decision trees). The results of the experiments are presented in 6.3. While performances of deep models are still above the

|                    | F-Beta    |          | N-Way     |          |
|--------------------|-----------|----------|-----------|----------|
|                    | Train set | Test set | Train set | Test set |
| Word-count DT      | 0.92      | 0.92     | 0.89      | 0.9      |
| PCA distances DT   | 0.87      | 0.90     | 0.71      | 0.69     |
| Multimodal DT      | 0.87      | 0.96     | 0.93      | 0.94     |
| Siamese BiLSTM     | 0.936     | 0.941    | 0.866     | 0.861    |
| Siamese CNN        | 0.969     | 0.976    | 0.969     | 0.956    |
| Multimodal Siamese | 0.991     | 0.989    | 0.984     | 0.978    |

TABLE 6.3: Performance comparison between Deep Models and Decision Trees

decision tree algorithms, there is no denying that such a simple algorithm using such simple features is impressively efficient. Hence, simpler Machine Learning models should not be disregarded by E.Fundamentals for this task. These very good performances are signs of the quality and potential of the dataset. Indeed, the main strength of the Deep Learning algorithm, its invariance to change, is not as crucial here as product pictures are, in a way, standardised: the front of the packaging is shown, centred on the image with a white background.

Furthermore, when investigating the PCA components, and transforming them back into images, we obtain the results displayed in Figure 6.4. The images resulting from this transformation are showing how sensible the PCA components are to some typical shapes: a square/rectangular shape centred in the image. On some specific maps, we see how the eigenvalues capture different brands of product. An example is *Dove*, to choose only one, which seems to appear in multiple of these images (the 4th, 6th and a part of its D in the 12th, reading left to right and top to bottom). This is also a sign of the low variance of the data, that results in PCA components that can efficiently reduce the dimension of the input data without a lot of loss of information, confirming the hypothesis that the very good performance of the Decision Trees is directly linked

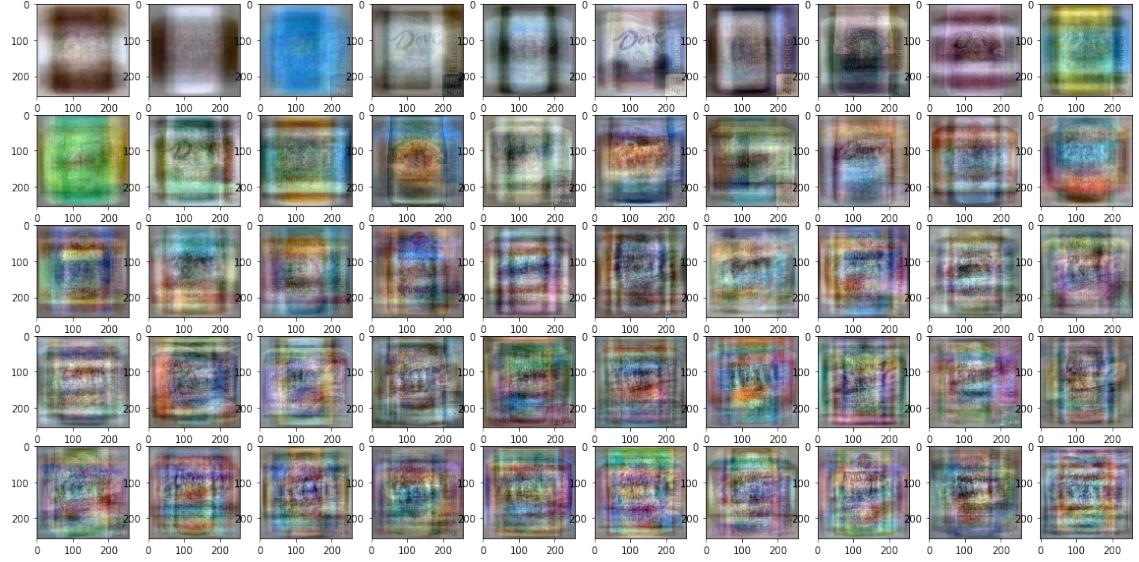


FIGURE 6.4: PCA Components reshaped as pixel maps

to the representational capabilities of the PCA, themselves a sign of quality data. It may then arise the question of the behaviour of these algorithms on harder dataset.

### 6.2.2 RQ2: *To what extent does having a siamese multimodal model give better performance than unimodal models? What is the impact on the resulting embedding ?*

Considering the research question N° 2, we suggested the following hypothesis:

*“If a multimodal model is used then it will show better performance than unimodal models as it will capture and extract features across multiple modalities at once, impacting the embedding”*

To verify this hypothesis, we will split the research question in two: first, we will analyse performance differences between modalities and then, we will explore the impact on the resulting embedding.

#### 6.2.2.1 Deep Learning model performance comparison

The table 6.4 acts as a reminder of the siamese neural network performances displayed in RQ1.

|                    | F-Beta    |          | N-Way     |          |
|--------------------|-----------|----------|-----------|----------|
|                    | Train set | Test set | Train set | Test set |
| Siamese BiLSTM     | 0.936     | 0.941    | 0.866     | 0.861    |
| Siamese CNN        | 0.969     | 0.976    | 0.969     | 0.956    |
| Multimodal Siamese | 0.991     | 0.989    | 0.984     | 0.978    |

TABLE 6.4: Performance comparison between Deep Models

**Analysis of title processing performance** In term of performance, we see that the models exploiting the image information are more robust to the N-way testing. Indeed, the Siamese BiLSTM seems to fail in that case. A hypothesis for this lower performance may be linked to what the LSTMs focus on when making their similarity prediction. It could logically be if they rely too much on the brand to compute similarity: if two products have the same brand, then our LSTM might consider these two products as similar. As it works on sequential data and outputs a value for each sequence, we can explore the activations of the BiLSTM. Then, the highest activation of the last LSTM amongst every timestep will point out which word the model put more focus on.

We see clearly in this heatmap (cf. Figure 6.5) how often is the first word of a product title crucial to determining similarity. When computing this activation over the whole dataset, in almost 62% of the time, the highest activation is on the first word of the input sentence, which contains the brand, or a part of the brand, the most of the time. Hence, if too much importance is given to the brand, then in an N-Way situation, it seems logical that the BiLSTM would tend to fail more often than the other two models as there is a high chance that within the  $N - 1$  negative images, at least one will be from the same brand as the anchor product title. When computing the same heatmap for the BiLSTM learnt using the multimodal siamese network, we obtain the Figure 6.6.

The captivating results of this heatmap lie within how different the focus word of the LSTM is: it looks like the brand is disregarded and that most of the attention is set on characteristics of the product (quantity, weight, type). In comparison to the Siamese RNN, having the highest activation of the first word happens only in 17 cases out of 4193, which is just around 0.4% of the time. We may see here the first sign of collaborative training between modalities: while the task of recognising if two products are from the same brand is easy for the Convolutional Neural Network, focusing on quantity and such detail labels may be harder and could then be left to the LSTM that processes the Product Title for which this task is way easier. We will now try to dive deeper into this collaborative learning procedure to uncover more details of how much modalities

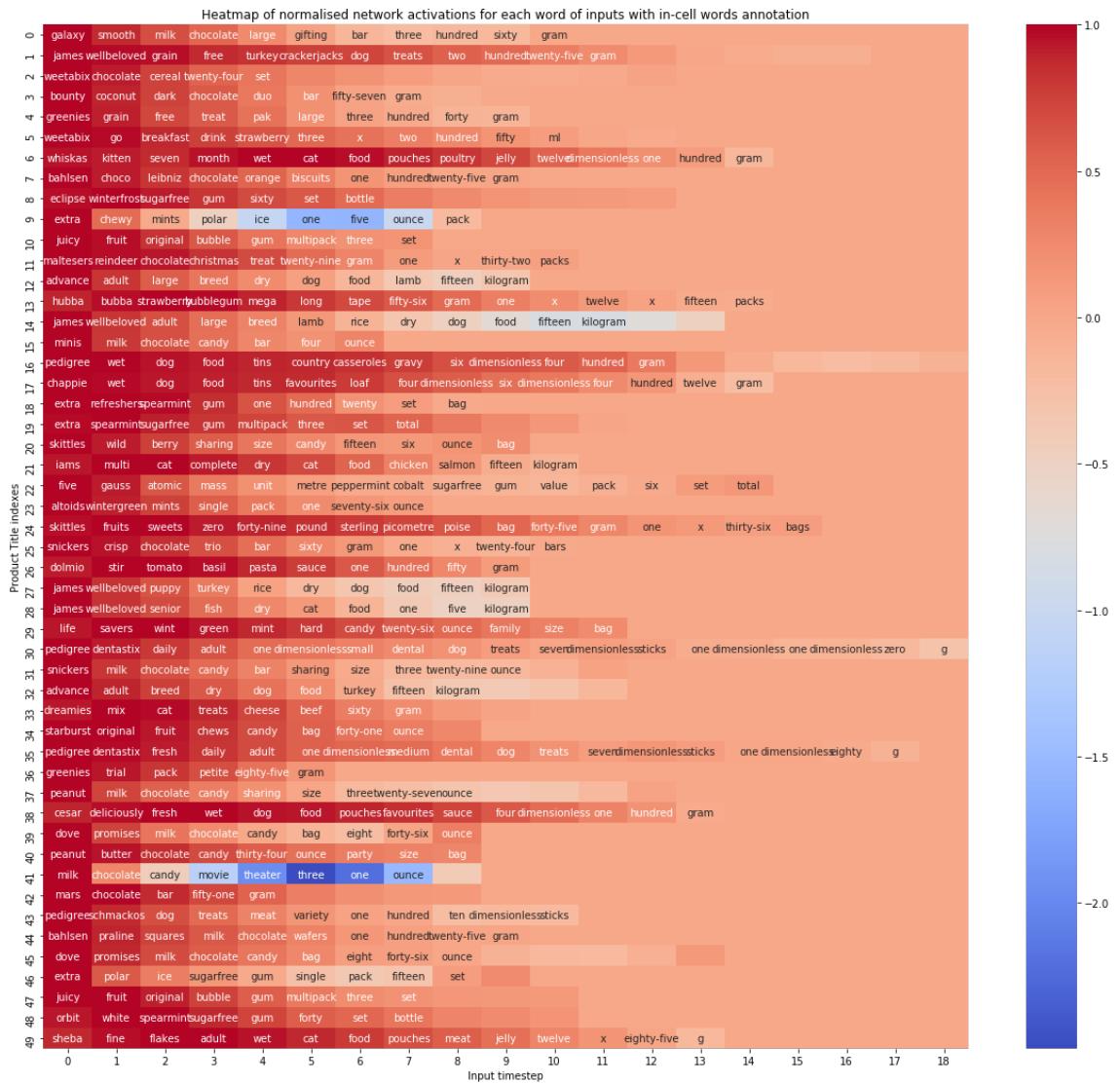


FIGURE 6.5: Heatmap of the normalised LSTM output activations from the Siamese Recurrent Neural Network

collaborate for this task.

**Embedding exploration & Multimodal collaboration** Provided that modalities do collaborate, as we have started to uncover above, maybe can we also notice such behaviour in the final embeddings of these 3 networks ?

To see how different they may be, we will project the high-dimensional embeddings in a 2D dimension by plotting the result of the application of a t-SNE algorithm on the embedding outputs of these 3 networks, as well as on the input image space to see any evolution.

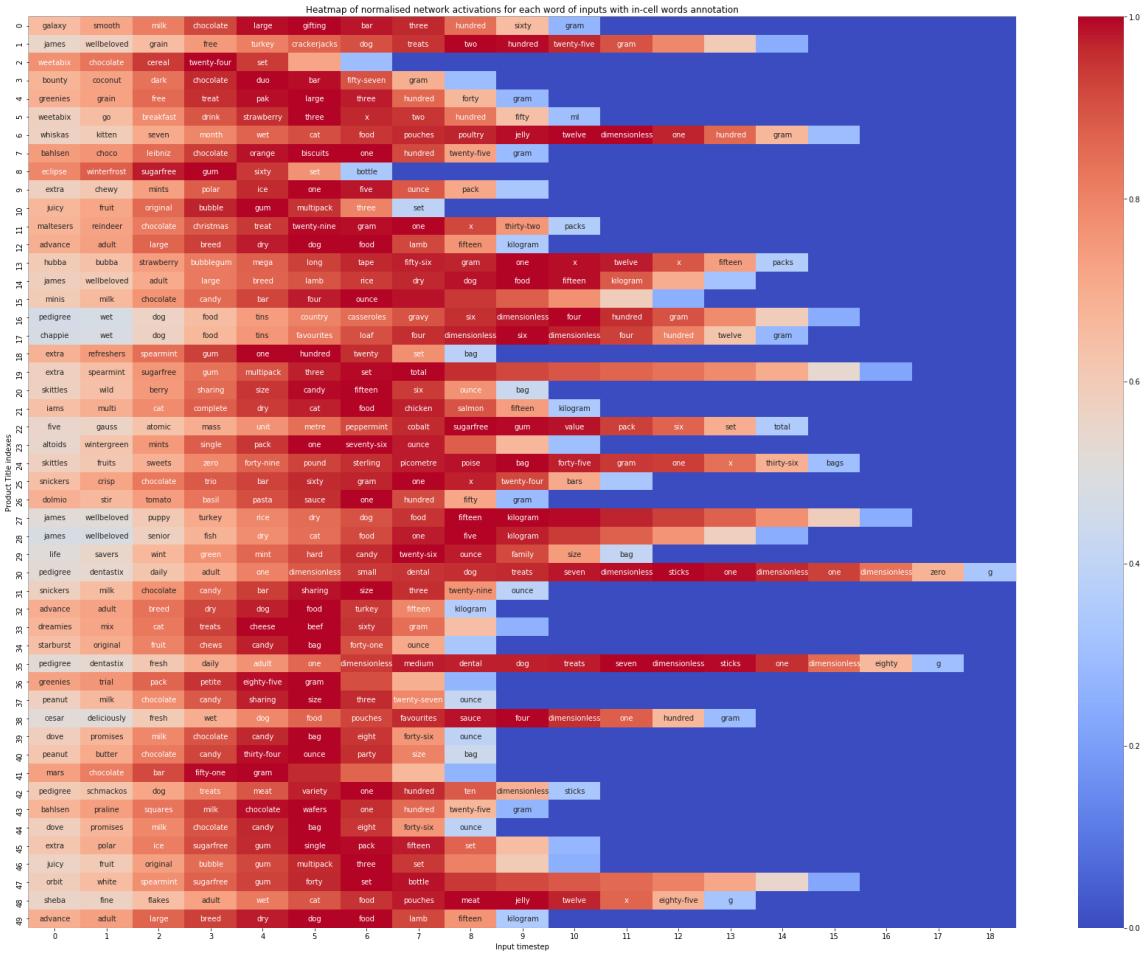


FIGURE 6.6: Heatmap of the normalised LSTM output activations from the Siamese Multimodal Neural Network

As presented in Figure 6.7, we see a very different degree of separation of products: the most scattered embeddings seem to originate from the Siamese RNN (N° 2). However, it does look like it clusters products by brands, putting closer together almost similar yet dissimilar products. As we had seen before, it does not seem surprising given the properties we uncovered when exploring RQ1.

However, it raises interesting questions with regards to the influence of the RNNs scattered embeddings on the multimodal embeddings (N° 4), compared to the embeddings of images (N° 3). In Siamese CNN embeddings, products do not appear especially scattered, apart from a few groups gravitating around the core of the plot. Despite this, we can observe that products are mostly arranged based on their shape while still being closer to similar other products.

The addition of the textual embedding to this plot can be seen in Nº 4: in this multimodal embeddings plot, scatteredness is increased. And it seems to find a very good

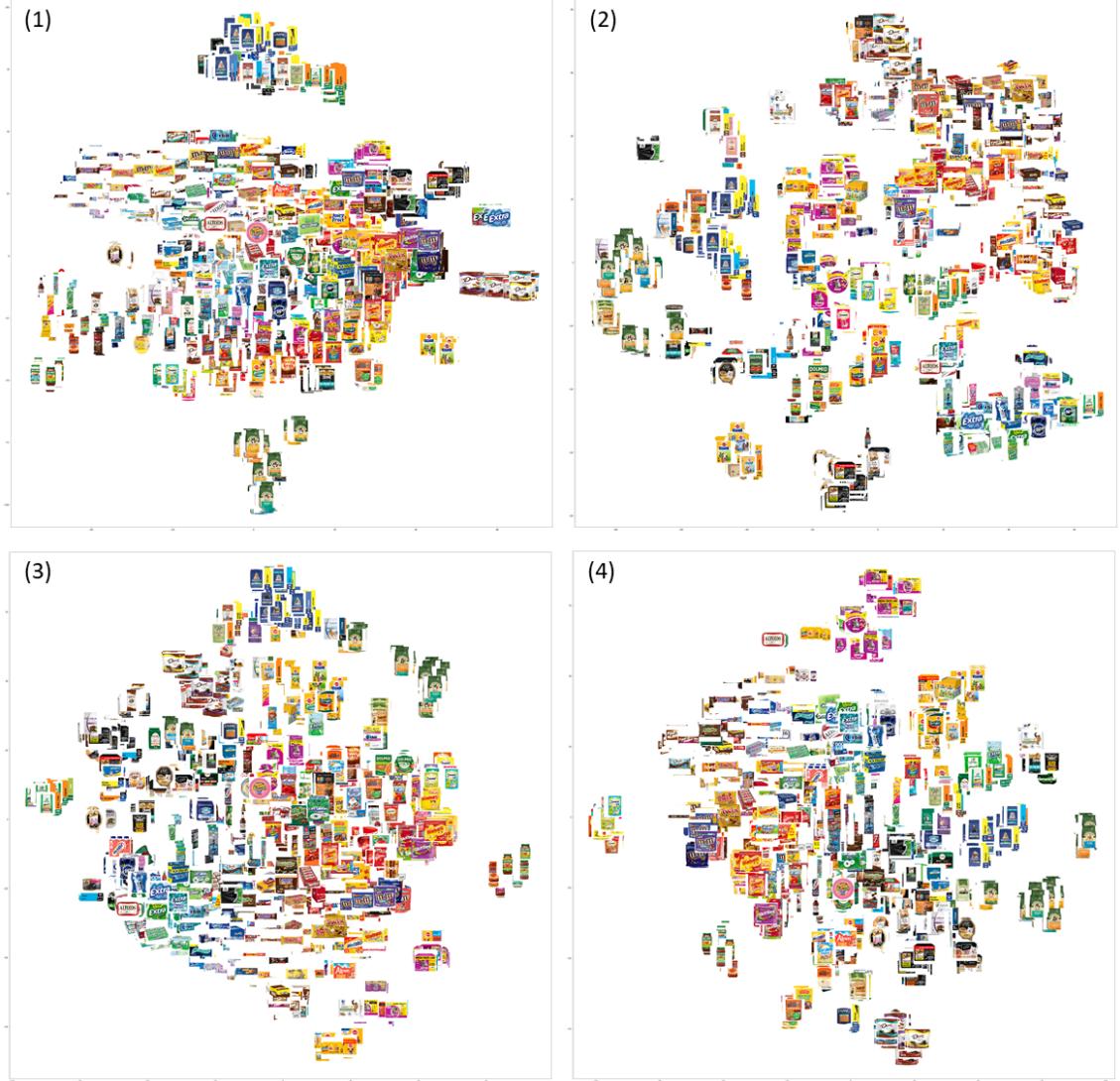


FIGURE 6.7: Plots of t-SNE's representation of 4 embeddings

(1) PCA-Reduced image data, (2) Siamese RNN embeddings<sup>4</sup>, (3) Siamese CNN embeddings, (4) Multimodal Neural Network embeddings

“balance”. Indeed, it seems to be a mix between the scatteredness properties of plot N° 2 with the addition of more accurate semantic clustering from plot N° 3.

These results seem to also validate the multimodal collaboration hypothesis. To conclude on this, we can take advantage of the weighted multi-modality merging setup for the Multimodal Siamese Network by observing the learnt weights of this network. These weights highlight the contribution of each modality to the final decision of the

| LSTM Merging weight | CNN Merging weight |
|---------------------|--------------------|
| 0.0828              | 0.1640             |

TABLE 6.5: Table of multimodal weights of the merging layer

multimodal neural network. Being relatively balanced, in an order of 1/3 and 2/3, these weights are another proof suggesting that the two modalities do collaborate and that we do not see one take other the over.

### 6.2.3 RQ3: *How much does the Convolutional Neural Network's weights of the multimodal model differ from the weights of the same model but within the unimodal architecture ?*

Considering the research question N° 2, we suggested the following hypothesis:

*“If a siamese model uses multimodal data to train its image feature and text feature extractors, then it will find different visual and textual features by developing different weights.”*

We can relate this hypothesis to our findings in RQ2 with regards to the differences in weights of the BiLSTM architecture between the Siamese RNN and the Siamese Multimodal model. It may be possible to extrapolate the idea of collaborative learning between modalities to the image features extracted by our two convolutional networks: the Siamese CNN & the Siamese Multimodal Network.

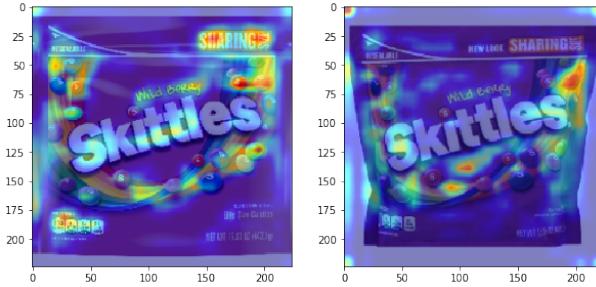
Both models are initialised with the same weights, pre-trained on the ImageNet dataset [Russakovsky et al., 2015]. When computing layer-wise weights difference, we obtain the distance values shown in Table 6.6. The column for the layer number references the indexing used in Listing 5.1. Comparing the weights distances to the average weight value, we notice a difference of proportion of about  $1e - 4$ , which appear to be a sign of little to no differences in comparison to the usual weight values. Hence, this may be the first sign that the learnt features are similar. In support of this first assumption, we will review weights activation of the different neural networks on similar images, in the same logic that we used in RQ2 when analysing word importance in the predictions of the RNN architectures.

To analyse the importance of the different image regions for the final similarity predictions, we will use the Grad-CAM tool [Selvaraju et al., 2016]. As described by the authors of this method, Grad-CAM “uses the class-specific gradient information flowing into the final convolutional layer of a CNN to produce a coarse localisation map of the

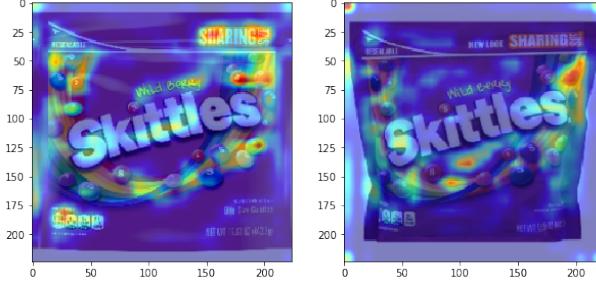
| Layer # | Weights distance | Mean weights value & variance |
|---------|------------------|-------------------------------|
| 0       | $3.7e - 0.8$     | $-1.9e - 3, 6.3e - 2$         |
| 2       | $1.6e - 07$      | $-1.7e - 3, 3.1e - 3$         |
| 5       | $1.6e - 07$      | $-1.8e - 3, 2.5e - 3$         |
| 7       | $1.9e - 07$      | $-2.2e - 3, 1.5e - 3$         |
| 10      | $1.9e - 07$      | $-1.3e - 3, 1.0e - 3$         |
| 12      | $1.4e - 07$      | $-1.4e - 3, 6.0e - 4$         |
| 14      | $1.2e - 07$      | $-2.4e - 3, 6.3e - 4$         |
| 17      | $1.7e - 07$      | $-1.7e - 3, 5.3e - 4$         |
| 19      | $2.0e - 07$      | $-1.6e - 3, 3.3e - 4$         |
| 21      | $1.8e - 07$      | $-2.2e - 3, 3.3e - 4$         |
| 24      | $2.6e - 07$      | $-1.6e - 3, 3.3e - 4$         |
| 26      | $2.7e - 07$      | $-1.8e - 3, 3.3e - 4$         |
| 28      | $2.1e - 07$      | $-2.2e - 3, 3.3e - 4$         |

TABLE 6.6: Layer-wise weights difference

*important regions in the image*”. The generated heatmap from this method helps to better understand the network’s decisions. In Figure 6.9, we display a comparison of



(A) Grad-CAM heatmaps of the similarity prediction of the Siamese CNN, with activation focus set on layer 21



(B) Grad-CAM heatmaps of the similarity prediction of the Siamese Multimodal Network, with activation focus set on layer 21

FIGURE 6.9: Comparison of the Grad-CAM of the activation maps on a similarity task for the Siamese CNN (A) and Siamese Multimodal (B) models

these activation heatmaps during similarity computation for the input images. We see here no noticeable difference, demonstrating similar results than for the layer weights comparison. Additionally, when directly looking at the weights of both models and the

non-finetuned pre-trained weights of the VGG16 model, we see similar results of very little to no changes in weights, This may be a sign of two things:

- ImageNet Pretrained Weights extracts features that are sufficient for the task, meaning that gradient disappear very early in the layers and no training is done at this stage, meaning that only the embedding layer is learnt, transforming the convolutional output into a 256-sized vector used for distance calculation;
- Gradient stuck in a local optimum: the sampling method may not be selecting hard enough examples, the optimizer may propagate gradient in an ideal manner... Multiple explanations may justify this low-training behaviour with regards to the convolutional layer weights.

I suppose the second theory to be more plausible than the first: while ImageNet features from the first layers may not need any tuning, there are most certainly improvements to be made in the most complex convolutional layers where combinatorial operations of low-level features result in high-level features, which are almost always specific to the task. This may be a symptom of a phenomenon we already observed when exploring RQ1 with regards to the input data: the data set has clean properties that have been captured by a simple PCA transformation (cf Figure 6.4) where we see it register within its components highly complex features such as brand title design. This demonstrates that if the task of the neural network is not hard enough, the backpropagated error signal might not trigger the expected weight updates.

A way to tackle this issue is through multitasking, as seen in 2.4.3.2. To explore this idea, we will analyse the 4<sup>th</sup> hypothesis.

#### 6.2.4 RQ4: *Can multitask learning improve model performance ?*

Considering the research question N° 4, we suggested the following hypothesis:

*“If a siamese neural network is trained or pre-trained using multitask learning then it will have better performance and generalisation than another model trained without multitasking.”*

While multitask learning presented by Wang and Yao [2019] accumulates gradient from multiple tasks in a parallel manner, a similar experiment could not be run due to computation limitations. However, Deep Learning is known for its excellent properties of iterative learning and the experiment was designed differently while following the key principles of multitask learning developed by Wang and Yao [2019].

For that matter, the procedure illustrated in Figure 6.10. In this method, we introduce

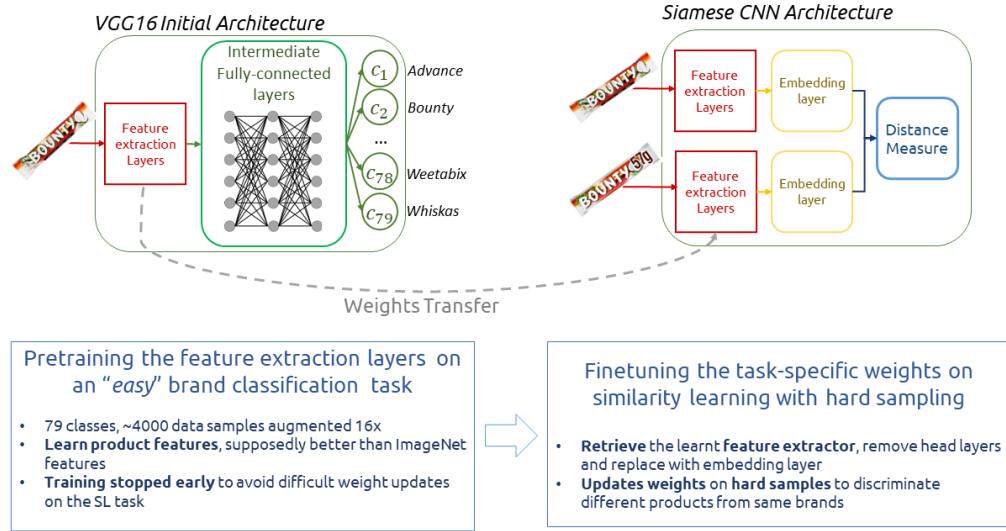


FIGURE 6.10: Multi-Task training procedure

a second task believed to help the network to learn features coherent with the current dataset. This task is to predict the brand of the product based on the image input. This method follows a simple list of instructions to follow:

1. Initialise a **VGG16** architecture with **ImageNet** pretrained weights;
2. **Augment** the base **dataset** by a given factor (in the presented experiment, a factor of 16 was used);
3. **Train** the initialised **VGG16** on the task of **predicting brand** based on product picture;
4. **Interrupt training** when performance are considered “*good enough*”. No precise criterion is given for that matter and empirical tests might need to be carried out. In the current experiment, the training was stopped when the F-Beta classification score exceeded 0.8 with  $\beta = 0.2$ ;

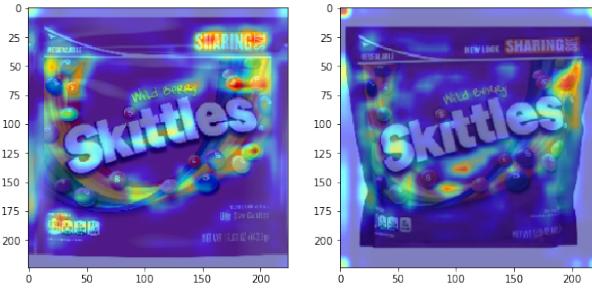
5. Transfer the **feature extraction** block of the VGG16 architecture to a **Siamese CNN** architecture;
6. Train & evaluate the **Siamese CNN** architecture on the **similarity learning task**.

When following this procedure, by training the Siamese CNN with the same conditions as presented in 6.1.2.2, the results displayed in Table 6.7 show significant improvement over the Siamese CNN architecture.

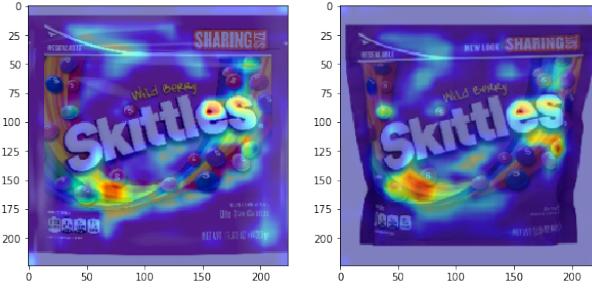
|                               | F-Beta       |              | N-Way        |              |
|-------------------------------|--------------|--------------|--------------|--------------|
|                               | Train set    | Test set     | Train set    | Test set     |
| Siamese BiLSTM                | 0.936        | 0.941        | 0.866        | 0.861        |
| Siamese CNN                   | 0.969        | 0.976        | 0.969        | 0.956        |
| <b>Pretrained Siamese CNN</b> | <b>0.968</b> | <b>0.974</b> | <b>0.970</b> | <b>0.976</b> |
| Multimodal Siamese            | 0.991        | 0.989        | 0.984        | 0.978        |

TABLE 6.7: Performance comparison between Deep Models

In Figure 6.11b, we can observe the Grad-CAM of predicted objects where differences in activation are noticeable.



(A) Grad-CAM heatmaps of the similarity prediction of the Siamese CNN, with activation focus set on layer 21



(B) Grad-CAM heatmaps of the similarity prediction of the pretrained Siamese Convolutional Network, with activation focus set on layer 21

FIGURE 6.11: Comparison of the Grad-CAM of the activation maps on a similarity task for the Siamese CNN (A) and the PreTrained Siamese CNN (B) models

Through these results, we show evidence that employing a multitasking training strategy, even without parallelising the training and sharing the gradient through multiple tasks. This pretraining of a siamese neural network forces it to learn new features, relevant to the task of identifying similar products and, as displayed in 6.7, improves n-way classification performance.

### 6.3 Conclusion

Throughout this chapter, we detailed experimental design and as well as the results of these experiments. At first, we studied the context of these experiments, how they were carried out, with which settings and parameters that may influence reproducibility. Then, we presented the 4 research question that focusing on multiple research points:

- The potential of Siamese models compare to traditional ML approaches (RQ1);
- The interactions between the learnt embeddings of different modalities at training time (RQ2, RQ3, RQ4);
- The influence of task modelling on the resulting learnt weights (RQ2, RQ3, RQ4).

By exploring these 3 points, we can conclude that multimodal siamese models have a lot of potential for the task of similarity learning, that combining multiple modalities in the same embedding improves performance, interpretability and has an impact on the learnt weights. Finally, we saw how multitasking was able to improve the performance of a Siamese CNN by learning new features that better fit the task.

This may then raise questions such as:

- Why were weight updates minimal when starting from ImageNet pre-trained VGG16 weights? Were we in a case of “*vanishing gradient*” or was the error minimal from the start?
- How would online multitasking influence the learnt features? How do they compare to the features of the pre-trained siamese CNN?
- Would a bigger batch size and then harder sampling improve the performance of the different siamese algorithms?

# Chapter 7

## Conclusion & Future Work

### 7.1 Conclusion

As a data company, E.Fundamentals retrieves a lot of information about products from multiple retailers daily. Being able to safely link identical products across retailers is their objective and this is what this MSc project was about: developing a Deep Learning tool that can detect product similarity across multiple modalities.

For that intent, different solutions were developed and 4 were presented in this report:

- One multimodal Machine Learning using PCA to extract image features and word count to compare product titles;
- 2 Unimodal Deep Learning models using the scheme of Siamese Neural Network: they duplicate the feature extraction and embedding layers for their two inputs before computing and classifying their distance;
- One multimodal Deep Learning models also following the scheme of Siamese Neural Networks but across multiple modalities: these modalities were processing in parallel through corresponding feature extraction layers before merging their output into a common embedding that could then be used to compute distances between multiple data samples;

All solutions developed were successful to multiple extents. Through experimentation, a performance boost of unimodal models was seen when using Multimodal Siamese

Networks with an embedding layer weighting and merging the multimodal input embeddings. This shows how different modalities are well managed and merged by Deep Neural Network that can then learn common concepts shared across texts and images. Furthermore, the interactions between embeddings at train time were also explored and it was proven how training multiple modalities on parallel could be impactful on their mutual extracted features. This collaborative training across modalities arose multiple questions with regards to the details of the said interactions. Finally, we explored another area of deep embedding learning with the exploitation of multiple tasks to learn more generalizable features which have significantly improved N-Way score of the Siamese CNN network.

## 7.2 Future work & reflections

The task of multimodal similarity learning raises multiple questions with regards to both similarity learning and multimodal deep learning: this unique crossroad of concepts can help to better understand every one of them unilaterally. Indeed, with regards to my models performance and the potential of improvement they have, there still exists a lot of open questions: What would be the impact of the batch size on training, especially with the use of mini-batch hard sampling ? What would be the training performance of our model in a parallel multitask learning situation compared to the sequential pre-training/training setup ? Could the use of a stack of unshared and shared feature extraction layers improve model performance by capturing potential distribution differences between the “*true products*” and the “*extracted products*” ? If such a profile can be identified, it could potentially benefit E.Fundamentals to extract cleaner data from retailers websites.

Outside of the context of E.Fundamentals, the use of multimodal Siamese Learning can be crucial for tasks where labelling is minimal and classes are in high number: for example, in the context of Remote Sensing, the use of multiple sets of modalities is very frequent. Capturing satellite imagery using different type of sensors can measure different type of information. For example, a combination of optic and SAR imagery can help to better evaluate land cover (e.g. [Gaetano et al., 2017]). In such a context, a similarity learning algorithm could be used on a lower scale basis to monitor change over

time of land cover. With this idea in mind, Multimodal Siamese Neural Network can prove themselves very useful to capture cross-modality information.

## Appendix A

# Configuration File

This is the code of the custom Python configuration class I created to follow the hyperparameters of my experiments. The constants are the default parameters for the experiments. Comments explain the use of each of these parameters.

---

```
import json

BATCH_SIZE = 128
EMBEDDING_DIM = 20
EPOCHS = 10
TRAIN_SIZE = 0.8
TEST_SIZE = 0.1
VAL_SIZE = 0.1
SHUFFLE = True
SEED = 42
LR = 1e-3
IMG_SHAPE = (128, 128)
MARGIN = 1.0
BETA = 0.2
N = 16
FACTOR = 4

class Config():
    """
    Class to use to define constants of training (hyperparameters, paths)
    """

    def __init__(self,
                 batch_size = BATCH_SIZE,
                 embedding_dim = EMBEDDING_DIM,
                 epochs = EPOCHS,
```

```
    train_size = TRAIN_SIZE,
    test_size = TEST_SIZE,
    val_size = VAL_SIZE,
    shuffle = SHUFFLE,
    seed = SEED,
    lr = LR,
    img_shape = IMG_SHAPE,
    margin = MARGIN,
    beta = BETA,
    n = N,
    factor = FACTOR,
):

    self.batch_size = batch_size
    """ batch size variable """
    self.embedding_dim = embedding_dim
    """ defines the dimension for the text embedding vectors """
    self.epochs = epochs
    """ defines the number of epochs for training """
    self.train_size = train_size
    """ defines (in percentage) the size of the training set """
    self.test_size = test_size
    """ defines (in percentage) the size of the test set """
    self.val_size = val_size
    """ defines (in percentage) the size of the validation set """
    self.shuffle = shuffle
    """ defines if the dataset is shuffled """
    self.seed = seed
    """ defines the seed """
    self.lr = lr
    """ defines the learning rate for the optimizer """
    self.img_shape = img_shape
    """ defines the input image shape for the image classifier """
    self.margin = margin
    """ defines the margin for the contrastive loss """
    self.beta = beta
    """ defines the beta parameter for the F-beta scoring function. """
    self.N = n
    """ defines the number of products sampled when doing N-Way testing """
    self.factor = factor
    """ defines the augmentation factor, if data aug is activated"""

def __str__(self):
    """ converts the config class as a string for monitoring """
    return json.dumps(vars(self), indent=2)
```

---

## Appendix B

### Plot of Decision Trees

In this appendix, you will find the plot of the 3 different decision trees used in [6.2.1.1](#).

#### Text-Modality Decision Tree

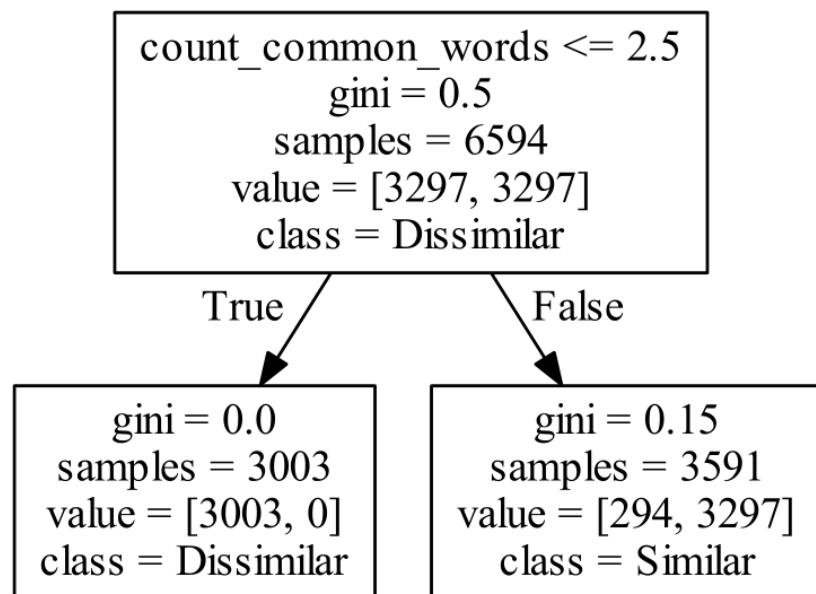


FIGURE B.1: Decision tree exploiting the word count feature

### Image-Modality Decision Tree

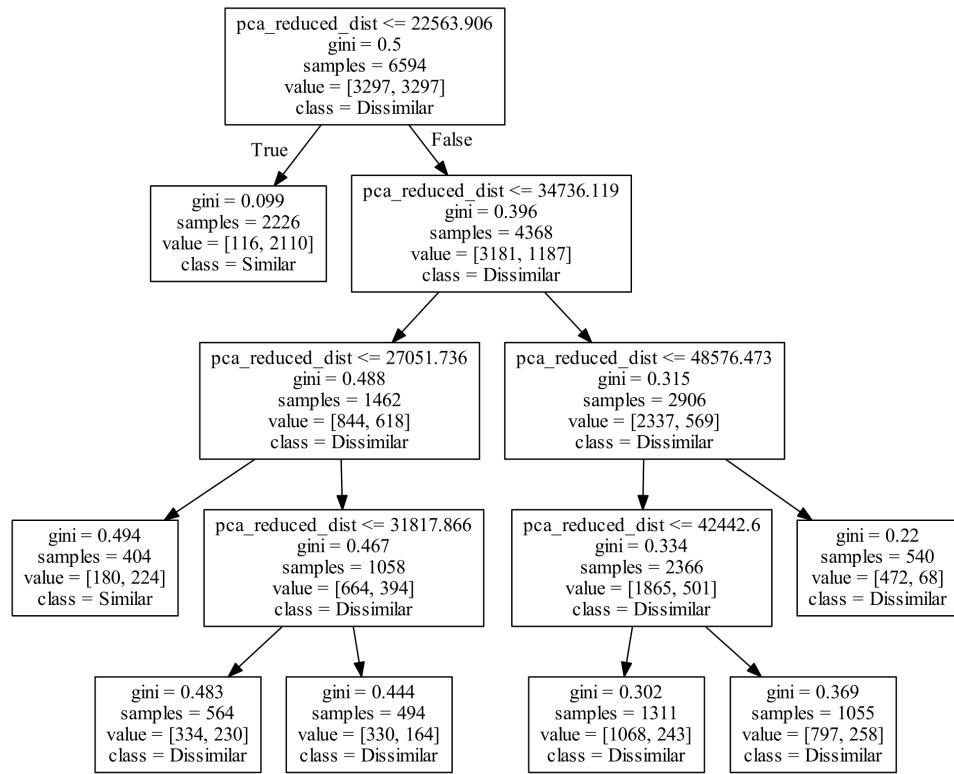


FIGURE B.2: Decision tree exploiting the image distance feature

### Multi-Modality Decision Tree

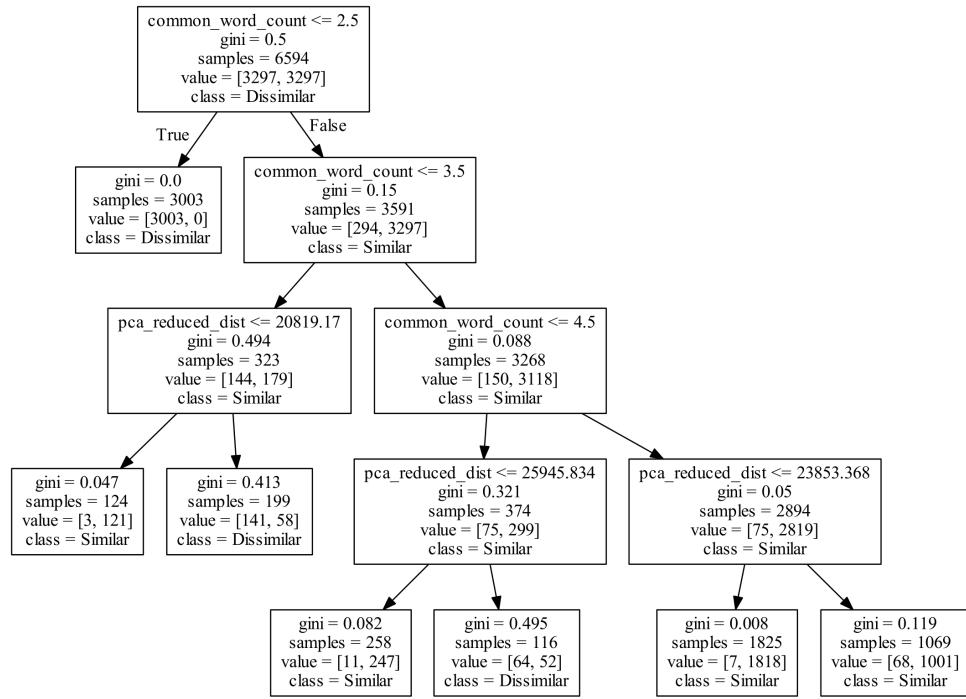


FIGURE B.3: Decision tree exploiting both features

# Bibliography

- Alexei, B. (2018). Performance metrics (error measures) in machine learning regression, forecasting and prognostics: Properties and typology. *ArXiv*, abs/1809.03006.
- Bertinetto, L., Valmadre, J., Henriques, J. F., Vedaldi, A., and Torr, P. H. S. (2016). Fully-convolutional siamese networks for object tracking.
- Christophe, C., Pavel, K., and Ladislav, L. (2017). On the effects of using word2vec representations in neural networks for dialogue act recognition. *Computer Speech Language*.
- Ciresan, D., Meier, U., and Schmidhuber, J. (2012). Multi-column deep neural networks for image classification. *2012 IEEE Conference on Computer Vision and Pattern Recognition*.
- Daudt, R. C., Saux, B. L., and Boulch, A. (2018). Fully convolutional siamese networks for change detection.
- Duan, Y., Chen, L., Lu, J., and Zhou, J. (2019). Deep embedding learning with discriminative sampling policy. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.
- Fukushima, K. (1980). A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36.
- Gaetano, R., Cozzolino, D., D'Amiano, L., Verdoliva, L., and Poggi, G. (2017). Fusion of sar-optical data for land cover monitoring. In *2017 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, pages 5470–5473.
- Geusebroek, J., Boomgaard, R. V. D., Smeulerds, A. W. M., and Geerts, H. (2001). Color invariance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

- Gregory, K., Richard, Z., and Salakhutdinov, R. (2015). Siamese neural networks for one-shot image recognition. *ICML Deep Learning workshop*.
- Hadsell, R., Chopra, S., and Lecun, Y. (2006). Dimensionality reduction by learning an invariant mapping. pages 1735 – 1742.
- Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization.
- LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *IEEE*.
- Lecun, Y., Haffner, P., and Bengio, Y. (2000). Object recognition with gradient-based learning.
- Li, H., Zhang, Q., and Sun, Z. (2017). Iris recognition on mobile devices using near-infrared images. *Human Recognition in Unconstrained Environments*.
- Li, M., Zhang, T., Chen, Y., and Smola, A. J. (2014). Efficient mini-batch training for stochastic optimization. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD ’14, page 661–670, New York, NY, USA. Association for Computing Machinery.
- Liu, W., Wang, Z., Liu, X., Zeng, N., Liu, Y., and Alsaadi, F. (2016). A survey of deep neural network architectures and their applications. *Neurocomputing*, 234.
- Nash, W., Drummond, T., and Birbilis, N. (2018). A review of deep learning in the study of materials degradation. *npj Materials Degradation*, 2.
- Nikolaus, M., Eddy, I., Philipp, F., Caner, H., Daniel, C., Alexey, D., and Thomas, B. (2018). What makes good synthetic training data for learning disparity and optical flow estimation? *CoRR*, abs/1801.06397.
- O’Shea, K. and Nash, R. (2015). An introduction to convolutional neural networks. *ArXiv e-prints*.
- Richard, S. (2010). Computer vision: Algorithms and applications. *Texts in Computer Science*, (1).
- Rosenblatt., F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*.

- Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. (2015). Imagenet large scale visual recognition challenge. *IJCV*.
- Schroff, F., Kalenichenko, D., and Philbin, J. (2015). Facenet: A unified embedding for face recognition and clustering. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.
- Selvaraju, R. R., Das, A., Vedantam, R., Cogswell, M., Parikh, D., and Batra, D. (2016). Grad-cam: Why did you say that?
- Sepp, H. and Jürgen, S. (1997). Long short-term memory. *Neural Computation*.
- Sherstinsky, A. (2018). Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network. *CoRR*, abs/1808.03314.
- Silberer, C. and Lapata, M. (2014). Learning grounded meaning representations with autoencoders. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 721–732, Baltimore, Maryland. Association for Computational Linguistics.
- Simeone, O. (2018). A very brief introduction to machine learning with applications to communication systems. *IEEE Transactions on Cognitive Communications and Networking*, 4(4):648–664.
- Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition.
- Sudip, K. and Hari Sagar, R. (2016). Comparison of a\*, euclidean and manhattan distance using influence map in ms. pac-man.
- Tomas, M., Kay, C., Greg, C., and Jeffrey, D. (2013). Efficient estimation of word representations in vector space.
- Wang, Y. and Yao, Q. (2019). Few-shot learning: A survey. *CoRR*, abs/1904.05046.
- Wehle, H.-D. (2017). Machine learning, deep learning, and ai: What's the difference?
- Wu, C.-Y., Manmatha, R., Smola, A. J., and Krähenbühl, P. (2017). Sampling matters in deep embedding learning.

Yamaguchi, K., Sakamoto, K., Akabane, T., and Fujimoto, Y. (1990). A neural network for speaker-independent isolated word recognition. *ICSLP 90*.

Zachary Chase, L. and John, B. (2015). A critical review of recurrent neural networks for sequence learning. *CoRR*, abs/1506.00019.