Automatic supervised information extraction of structured web data

Final Degree Project

REPORT

David Pérez Ruiz

Bachelor's degree in Aerospace Vehicle Engineering

Project director:

Alfredo Vellido Alcacena

June 2019

Terrassa, Spain

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONA**TECH**

Escola Superior d'Enginyeries Industrial, Aeroespacial i Audiovisual de Terrassa

I would like to thank the project director Alfredo Vellido for his help, support and flexibility during this work. I would also like to thank Mar and my familiy for putting up with me in these busy months. Pero en especial quiero agradecer y dedicar este trabajo a mi abuela Alicia, a quien le hubiera hecho muy feliz verme acabar la carrera.

Contents

List of Figures

State of the art

Since the revolution in information generation, processing and communication posed by the development and widespread take up of the World Wide Web, there have been attempts to extract information from user-oriented web pages. The first important attempts were made by Microsoft in 2003 [3], with their VIsion-based Page Segmentation (VIPS) algorithm, which divides content in the page in subsections and allows for easier processing of the data. The work in [4] was released that same year, and closes the scope by successfully extracting data from template-based web pages by guessing how information is filled in them (in what can be named as wrapper inference). It does not provide, though, a method for classifying the extracted data.

Another, more recent contribution in 2012 [5] proposes the use of user input to target certain type of data and, combined with wrapper inference and ontologies, has been proved to provide successful results. The latest advances in the use of Machine Learning in the form of Deep Learning can be found in 2016 [2], where a Convolutional Neural Network is applied to extract specific classes of information, thus solving the problem of categorizing the data through classification. Other recent works like [6] follow suit, and so does this work.

Abstract

The overall purpose of this project is, in short words, to create a system able to extract vital information from product web pages just like a human would. Information like the name of the product, its description, price tag, company that produces it, and so on. At a first glimpse, this may not seem extraordinary or technically difficult, since web scraping techniques exist from long ago (like the python library Beautiful Soup for instance, an HTML parser[1] released in 2004). But let us think for a second on what it *actually* means being able to extract desired information from *any* given web source: the way information is displayed can be extremely varied, not only visually, but also semantically. For instance, some hotel booking web pages display at once all prices for the different room types, while medium-sized consumer products in websites like Amazon offer the main product in detail and then more small-sized product recommendations further down the page, being the latter the preferred way of displaying assets by most retail companies. And each with its own styling and search engines. With the above said, the task of mining valuable data from the web now does not sound as easy as it first seemed. Hence the purpose of this project is to shine some light on the *Automatic Supervised Information Extraction of Structured Web Data* problem.

It is important to think if developing such a solution is really valuable at all. Such an endeavour both in time and computing resources should lead to a useful end result, at least on paper, to justify it. The opinion of this author is that it *does* lead to a potentially valuable result. The targeted extraction of information of publicly available consumer-oriented content at large scale in an accurate, reliable and future proof manner could provide an incredibly useful and large amount of data. This data, if kept updated, could create endless opportunities for Business Intelligence, although exactly which ones is beyond the scope of this work. A simple metaphor explains the potential value of this work: if an oil company were to be told where are all the oil reserves in the planet, it still should need to invest in machinery, workers and time to successfully exploit them, but half of the job would have already been done[2].

As the reader will see in this work, the way the issue is tackled is by building a somehow complex architecture that ends in an Artificial Neural Network[3]. A quick overview of such architecture is as follows: first find the URLs that lead to the product pages that contain the desired data that is going to be extracted inside a given site (like URLs that lead to "action figure" products inside the site ebay.com); second, per each URL passed, extract its HTML and make a screenshot of the page, and store this data in a suitable and scalable fashion; third, label the data that will be fed to the NN[4]; fourth, prepare the aforementioned data to be input in an NN; fifth, train the NN; and sixth, deploy the NN to make [hopefully accurate] predictions.

[1] Note that words "parsing" and "scraping" are used interchangeably.

[2] A more down-to-earth example could be the possibility of monitoring house prices across all major real estate companies, comparing the best value for each district, how prices fluctuate in time and location, availability, etc.

[3] Artificial Neural Networks are computing systems inspired by the biological networks in animal brains. They will be discussed later, but it is important to know that they are very useful at tasks like computer vision and speech recognition.

[4] NN will be used as abbreviation for Neural Network.

1 Introduction

In the following paragraphs, all components of this work will be presented to the reader. It is important to stress that the objective of this project is to try achieve a preliminary proof-of-concept to assess if the extraction of information from any web source and according to our requirements is possible. In fact, this extraction of information has been restrictively categorized to *name of the product* and *price tag* (with a third category stating *none* for neither product name or price tag). It is not expected to have a fully-fledged and deployable system, but rather an initial small architecture from which one can begin extracting conclusions: if it does work, how accurately? If it does not work, what are the reasons that make it fail? And most importantly, can they be fixed? If they can be fixed, how much time and computing resources are expected to be needed? These conclusions and future work are the true quintessence of the work. Underlying all this, there is a theoretical background surrounding Artificial NNs, for which some basic mathematical foundations concerning the architecture developed in this work will be provided in the last section of Methodology.

As mentioned in the Abstract, the project can be divided, from a technological viewpoint, into several subpieces or subprograms that do different jobs in the chain of extracting and processing data. In the following lines, they are briefly introduced in due order:

1. The *web crawler*. It is the software in charge of finding and indexing of the URLs that lead to product pages in a website. It should do so in an automated way and avoid being detected by the server (some servers detect they are being searched by a robot and fire countermeasures, which should be eluded).

2. The *extractor*. Given the URLs provided by the web crawler, the extractor should be able to process them and do two things: take a screenshot in a user-defined resolution and extract certain elements of its HTML, and conveniently store it.

3. *Labelling* the data. Since an NN is going to be implemented in further steps, there needs to be a proper influx of labeled[5] data to train it. Since the extracted data are unlabeled, a user-friendly platform is created to ease the process.

4. *Data cleaning*. Data must be cleaned and readied to be input to the NN. To such effect, a series of scripts are created.

5. *Training of the Artificial Neural Network*. And finally the creation of an Artificial NN [hopefully] capable of producing the desired results. It has an architecture of three inputs $\rightarrow$ one output, and is capable of being inputted data in batches. Its training is very slow (or even impossible when the computer runs out of memory) due to its computational workload, so a computation cluster is provided to run it. The accuracy of the NN in test data will be what determines the grade of success of the model and hence the project. Its architecture is based on the one described in [2].

[5]Labeled data means data the user will input to know its category and the truthful category it belongs to altogether.

2 Theory background

This section only emphasizes the Deep Learning aspect of the model architecture that has been developed for the project, since it is the part that holds most interest, importance and technical difficulty. In section 3, the specifics about its implementation are provided, while here the model is discussed from a lower-level point of view to understand what is "under the hood" when the deep learning software libraries are used. In this section, we discuss the inner workings of neurons as a system, and the functioning of the Convolutional Neural Networks that make use of them, from their architecture to their training methods. A final remark on pooling as a layer inside the grand scheme CNNs is also made.

2.1 The Convolutional Neural Network

Arguably, Convolutional Neural Networks are one of the architectures inside the world of Artificial Intelligence that is most inspired by neuroscience. As stated by [7] and [1], the history of convolutional networks begins with the collaborative research of neurophysiologists David H. Hubel and Torsten Wiesel. They widened the knowledge on the structure of the visual cortex experimenting with animals, showing that neurons in the visual cortex have a small *local receptive field* that strongly reacts to imagery inside its corresponding region of the visual field of the animal. They also discovered that receptive fields of neurons overlap and construct layers, where the lower-level ones react to simple patterns (like lines with different orientations) and the ones connected on top of them react to more increasingly complex patterns. These studies culminated in what today are called Convolutional Neural Networks, which belong to the family of Deep Learning methods.

2.1.1 Computation with neurons in general: the Multi-Layer Perceptron

A [simple] *perceptron* is based on a *linear threshold unit* (LTU). The LTU consists of one or more inputs with weights associated to each of them that converge to a weighted sum, and then applies a step function to it (commonly called activation function), as illustrated by Figure 1. The perceptron can be trained, and if increased in width (adding more LTUs) can solve some classification problems. Still, these systems are held back by important modelling limitations[6] whose discussion is beyond the scope of this work.

[6] Such as the XOR problem.

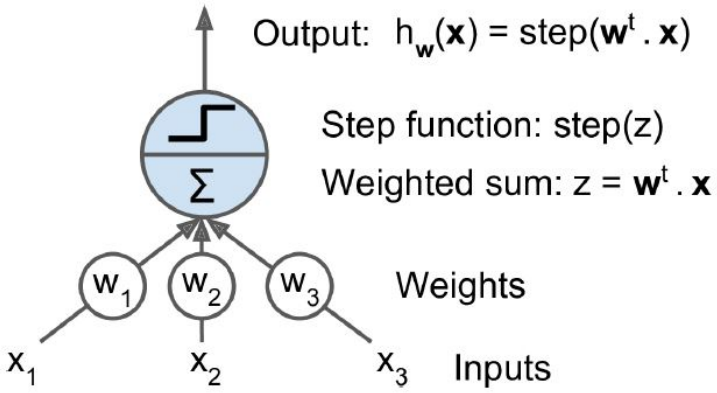

Figure 1: Perceptron of one linear threshold unit with three inputs. Figure extracted from [1].

The Multi-Layer Perceptron (MLP), as the name suggests, is composed of several layers[7] of perceptrons with several LTUs stacked one upon the other and with nonlinear activation functions (every layer of LTUs includes one bias neuron), and lacks the data modelling limitations perceptrons suffer. Still, the main difficulty with MLPs is how to successfully train them.

Modern MLPs like the one in 2 apply the *backpropagation* training algorithm. For each training instance input to the MLP the final output is computed and compared against the true value it should have. This is the training error (that is, desired output minus actual output obtained). Then it is measured how the outputs of the intermediary neurons contributed to said training error and adjusts the weights of their connections. The way it is measured is by reversing directionality (from output to input of the MLP), hence the name of backpropagation. The "adjustment" on the weights of neurons is usually a Gradient Descent optimization algorithm (explained in the Appendix). This step is repeated until the training cannot be improved (convergence), or until a generalization maximum is reached.

A trainable MLP should also change the step function inside every neuron by a differentiable function[8] to allow use of the Gradient Descent as the training function inside the backpropagation algorithm. The step function was initially changed by the logistic function, $\sigma(z) = 1/(1+exp(-z))$, but the most common activation function is the Rectified Linear Unit (ReLU) $h_{w,b} = max(X \cdot w + b, 0)$[9]. It is also the type of activation function used in this work.

[7]Note that when the word "layers" in plural is used, the Artificial NN is called a Deep Neural Network (DNN).
[8]Which therefore has a gradient instead of a flat segment like the step function.
[9]Where X is the input, w the weight and b the bias.

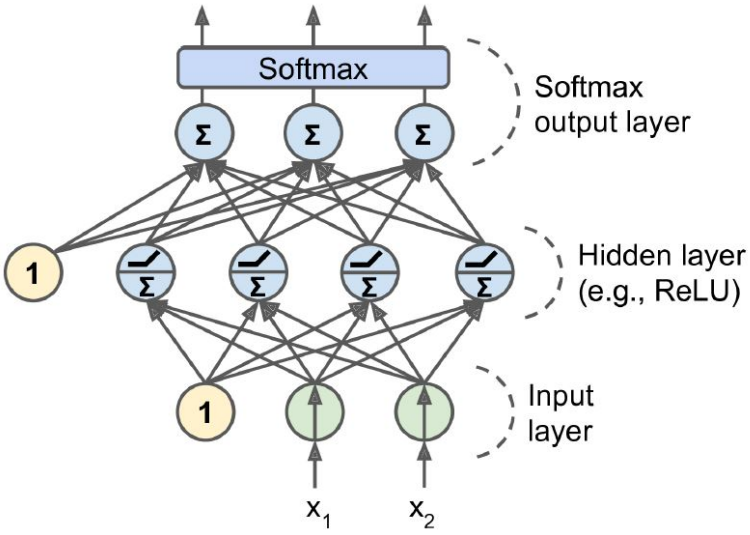

Figure 2: Multi-Layer Perceptron with Rectified Linear Units as activation and a Softmax layer as output. Figure extracted from [1].

The MLP also requires a last layer for the output, which could be multiple (multi-class problem). This is accomplished with a *softmax* layer, discussed in next subsection.

2.1.2 The softmax layer

The best way to explain the softmax layer (that is, a layer that applies the softmax function) is by first introducing the linear regression, which is quite straightforward and represents the mapping from input to output as a linear combination of the form:

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + ... + \theta_n x_n, \tag{1}$$

where $\hat{y}$ is the predicted value, n is the number of input features, x_i is the feature value (for example, if one wanted to predict the "level of success of a researcher", the input features could be the number of citations, the number of papers published and the number of conferences attended) and θ is the model parameter that multiplies each input feature (plus a bias term θ_0). Equation 1 can be easily vectorized as:

$$\hat{y} = \theta^T \cdot \mathbf{x}, \tag{2}$$

where θ^T is the parameters vector transposed (row vector) and $\mathbf{x}$ is the input feature vector (x_0 multiplied to θ_0 is always 1).

The softmax function is a nonlinear counterpart to equation 2, with the difference that it now supports several classes for every parameter vector. The output will therefore be a vector with as many values as classes, each one an estimation of the probability of the input value belonging to said class. The way this is done is by computing a score for each class, then computing its exponential and normalizing it. The score is calculated as:

$$s_k(\mathbf{x}) = (\theta^{(k)})^T \cdot \mathbf{x}, \tag{3}$$

where s_k is the softmax score for every class k and $\theta^(k)$ is the parameter vector for each class. Then:

$$\hat{p}_k = \frac{exp(s_k(\mathbf{x}))}{\sum_{i=1}^{K} exp(s_i(\mathbf{x}))}, \tag{4}$$

where K is the total number of classes k, is the estimated output for each class.

Given the softmax function in 4, its classifier prediction takes the form of:

$$\hat{y} = \underset{k}{argmax} \;\; \hat{p}_k \tag{5}$$

Equation 5 thus returns the probability value of the most probable k^{th} class. If, for example, p_k is a 2 cell vector with values [0.89, 0.11], then class 1 (and not class 2) will be the most probable class for a given input vector, with a probability value of 0.89. So far, the functions behind the workings of the softmax layer have been exposed. But, how to determine the θ^k parameters introduced in equation 3 to begin with? It is done by training the layer. How such thing can be done is explained in the following subsection.

2.1.3 Training of the softmax layer

The key concept to understand the training of the softmax layer is the *cost function*. To assess how well the model fits the training set, a performance measure must be used, and the most common one is the Mean Squared Error (MSE), described as:

$$MSE(\mathbf{X}, \theta) = \frac{1}{m} \sum_{i=1}^{m} (\theta^T \cdot \mathbf{x}^i - y^i)^2 \tag{6}$$

In the end, the objective is to compare the prediction delivered by the training parameters and the true prediction for that input instance. It has to be noted that the MSE cost function has a closed-form solution, but that is not the case for other cost functions. Optimization algorithms like Gradient Descent are needed, and those will also need the gradient of the cost function to work.

With the concept of cost function explained, now the cost function for the softmax function is presented: the *cross entropy cost function*[10] in equation 7. Note that Θ is the matrix of parameters:

$$J(\Theta) = -\frac{1}{m} \sum_{i=1}^{m} \sum_{k=1}^{K} y_k^{(i)} log\left(\hat{p}_k^{(i)}\right), \tag{7}$$

where K is the number of classes and m is the number of training instances in the dataset. In fact, equation 7 is the average cost function over the entire training set, hence the summation and the division by m.

Then, the *gradient of the cross entropy* function for a class k is defined as:

$$\nabla_{\theta^{(k)}} J(\Theta) = \frac{1}{m} \sum_{i=1}^{m} \left(\hat{p}_k^{(i)} - y_k^{(i)}\right). \tag{8}$$

[10] Yes, "cost function" was mentioned three times in just one sentence.

This equation computes the gradient for every class k, so it should be evaluated as many times as classes exist. The result of using a Gradient Descent optimization algorithm will be a parameters matrix Θ with as many rows as input features and as many columns as classes.

Now that the basics of Deep Neural Networks have been seen, the focus shifts towards the special type of architecture that is used in this work: the Convolutional Neural Network.

2.1.4 The convolution operation and its implementation

Let us suppose a 1D function $f(x)$ consisting on a series of delta functions along the axis, and another function $h(x)$ consisting on a Gauss continuous distribution. If f is convolved[11] with h, which will be called $g(i)$, then:

$$g(i) = (f \otimes h)(x) = \int_{-\infty}^{\infty} f(x)h(i-x)dx, \tag{9}$$

as illustrated in figure 3. This particular example is inspired by a video series of the California Institute of Technology[12].

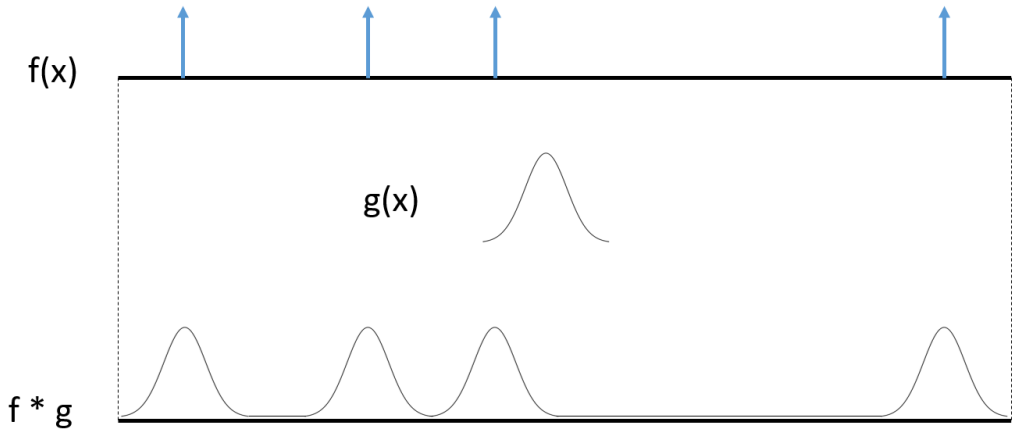

Figure 3: Delta functions distributed along an axis, $f(x)$, and a Gaussian distribution, $g(x)$. The convolution of f with g produces the function $f * g$, where the Gaussian is applied to every point in which $f(x)$ has the delta functions.

A better way to represent the utility of the convolution is to exemplify it in 2 dimensions, which is the type CNNs usually deal with. Let us take the case of a finely pixelated image[13] like that in Figure 4 left. If the image, thought of as a 2D matrix, is convoluted with a Gaussian spreading function, the result is a blurred image like the one in 4 right. The mathematical rationale behind it is:

[11] Denoted by an asterisk, although the symbol $\otimes$ is also used.
[12] https://www.youtube.com/watch?v=MQm6ZP1F6ms
[13] Source: https://en.wikipedia.org/wiki/Convolution

$$g(i,j) = (f \otimes h)(x) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x,y)h(i-x, j-y)dxdy \tag{10}$$

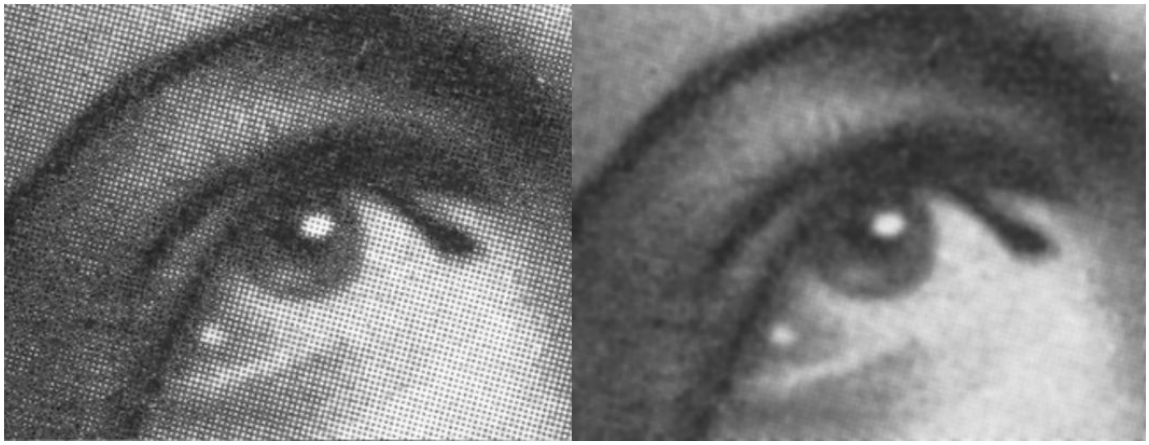

Figure 4: Finely pixelated image on the left, and after gaussian blur on the right.

But how does this translate into a computational scheme? There needs to be a discretization. For Eq. 9, it is:

$$g(i) = (f \otimes h)(i) = \sum_{x} f(x) * h(i-x), \tag{11}$$

while for Eq. 10:

$$g(i,j) = (I \otimes K)(i,j) = \sum_{m} \sum_{n} I(m,n)K(i-m, j-n) \tag{12}$$

In the "convolutional network" terminology, as stated in [7], $f(x)$ is considered the *input* (I) and $h(x)$ the *kernel* (K), and g, the output, is called *feature map*. Thanks to the commutative property of the convolution, Eq. 12 can be written as:

$$g(i,j) = (K \otimes I)(i,j) = \sum_{m} \sum_{n} I(i-m, j-n)K(m,n) \tag{13}$$

Eq. 13 is easier to implement because K tends to have the same range of values. From Eq. 13, it can be seen that as one index for the input increases, the index for the kernel decreases, and hence it can be said that said kernel is *flipped*. Most CNN implementations avoid using the mathematical operation of convolution altogether (even though they still use the term[14]) and use a very similar function called *cross-correlation*, which does exactly the same as the convolution does, without flipping the kernel. It is described as:

$$g(i,j) = (I \otimes K)(i,j) = \sum_{m} \sum_{n} I(i+m, j+n)K(m,n) \tag{14}$$

[14]Which is also the case of this work.

The graphic representation of Eq. 14 is displayed in Figure 5.

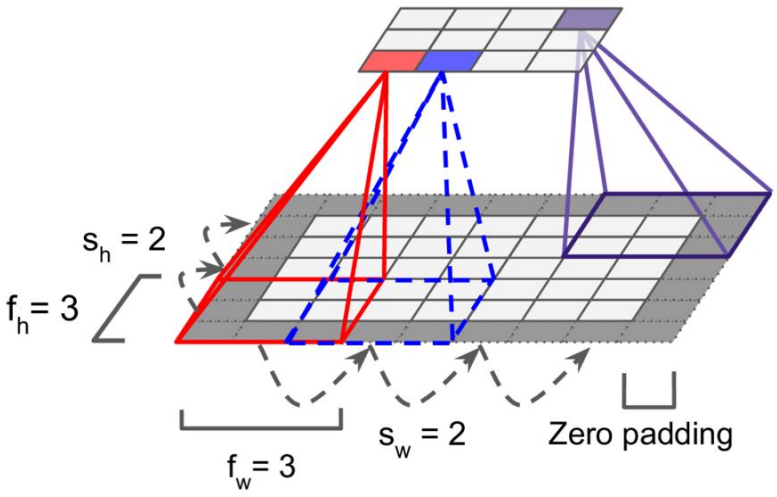

Figure 5: Image from [1] slightly edited. f_h and f_w stand for height and width of the receptive field, and can be understood as m and n of the cross-correlation function in 14. It can be seen that the size of the output layer is smaller than the input layer. That is caused by the *stride*, which is the distance from two consecutive receptive fields. Another important characteristic to point out is the existence of a zero-padding border surrounding the input. It is added to the input before the convolution to avoid the filter stepping out of bounds when it is being applied to the input. When stride = 1 the zero-padding also avoids shrinking the image, which allows an arbitrarily deep convolutional network.

2.1.5 A small remark on convolution kernels

Convolution kernels, also called filters, hold the neurons' weights and can be represented as an image the size of the receptive field. Filters are designed to detect features in the input, and can increase in complexity as layers increase. Figure 6, for instance, shows two possible filters that result in two different feature maps. The one on the left will multiply by [almost] zero all things except for vertical lines, while the filter on the right will ignore everything except for horizontal lines. That is why the feature map of the left has vertical lines brighter (enhanced) and the rest is blurred out. A similar thing happens for the one on the right. The point of training a CNN is to find the most useful filters for the task, and combine them.

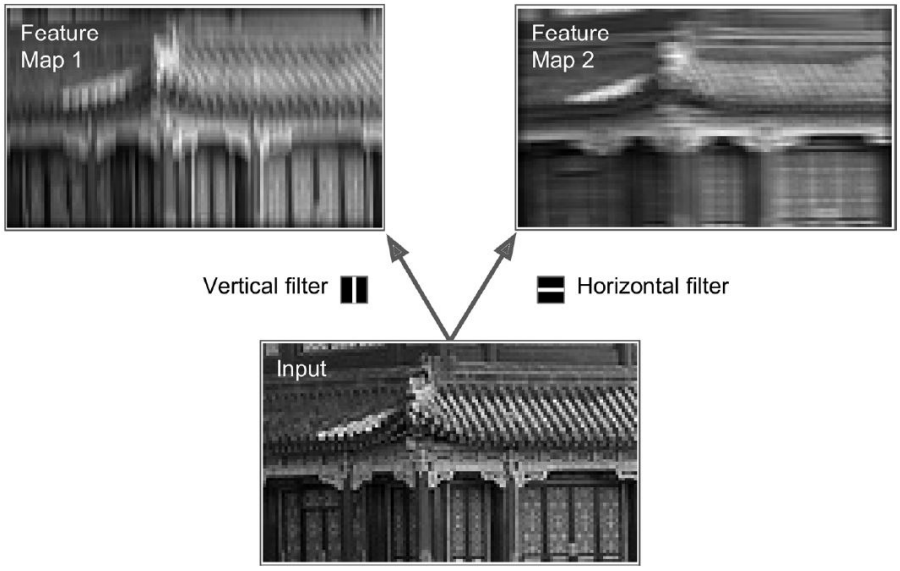

Figure 6: Image from [1]. Two different filters to detect vertical and horizontal features on the input image will generate two feature maps.

2.1.6 The feature maps

The use of different filters on the same input will inevitably lead to as many outputs as filters applied. The way this is handled is by stacking said outputs one on top of another, creating a 3D matrix. To that end, the same procedure is applied again (as many times as convolutional layers the user desires). Image 7 visually explains this process.

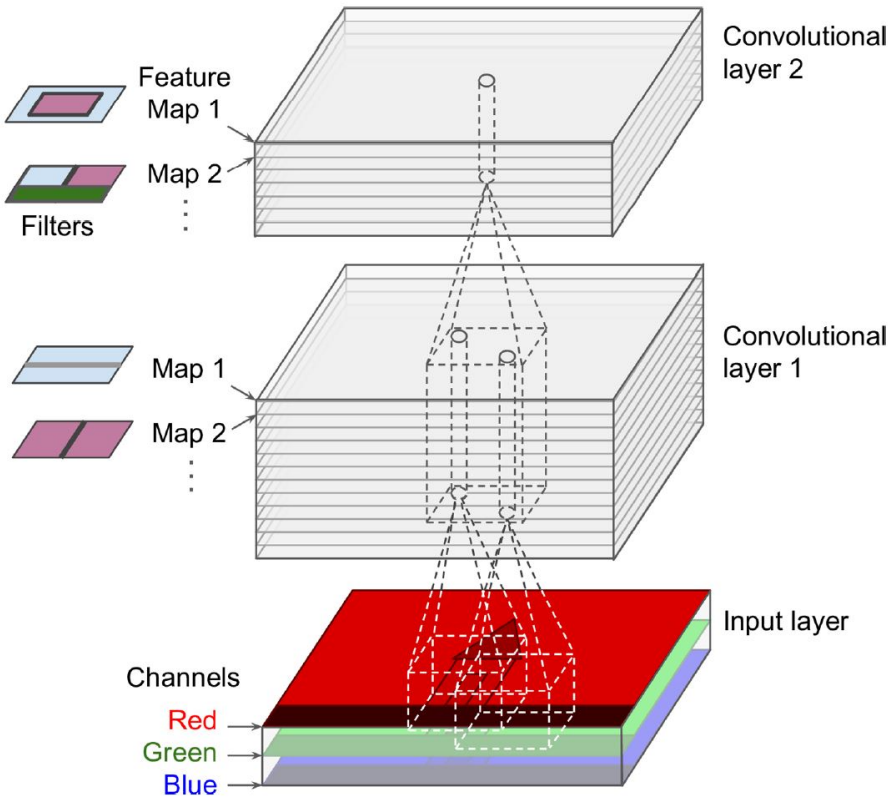

Figure 7: Image from [1]. Full color image represented by its three channels: Red, Green and Blue (RGB). A number of filters (kernels) are applied to the three channels, creating the same number of feature maps, which are stacked together. Higher-level convolution operations are applied to the previously stacked feature maps, with more complex filters to detect higher-level features.

2.1.7 More on training

In previous subsections, the training method for both the softmax layer and backpropagation in NNs in general have been exposed. In this subsection, a brief explanation of the actual training method used on this work is delivered: the *Root Mean Square back-Propagation* (RMSProp). The RMSProp algorithm was originally defined by Tijmen Tieleman and Geoffrey Hinton in 2012 for an educational video series for Coursera [8], and greatly outperforms GD.

RMSProp first calculates the accumulation of squares of gradients into vector $\mathbf{s}$, but doing so for most recent iterations using an exponential weighted moving average (explained in the Appendix). Then, a very similar function to GD is used with the addition of an element-wise division: $\sqrt{\mathbf{s}+\epsilon}$ [15]. This scales down the jump in the gradient vector and allows to tackle one of the GD's main weaknesses: it does not rapidly go down the steepest slope, but rather move more gently, and eases the reach of a global minimum. In fact, this is called an *adaptative learning rate*. Equations below show what has just been explained:

[15] ϵ is a small arbitrary value to avoid division by zero.

$$\mathbf{s} \leftarrow \beta\mathbf{s} + (1-\beta)\nabla_\theta J(\theta) \otimes \nabla_\theta J(\theta) \tag{15}$$

$$\theta \leftarrow \theta - \eta\nabla_\theta \oslash \sqrt{\mathbf{s}+\epsilon} \tag{16}$$

Apart from the RMSProp, other optimization algorithms are currently being used by the Machine Learning/Deep Learning community, such as Nesterov Accelerated Gradient, AdaGrad and Adam Optimization. Their explanation is, yet again, beyond the scope of this work.

2.2 The pooling layer

The pooling layer is quite simple. As the name suggests, its objective is to make the input smaller by grouping its values. It is very useful to reduce computational cost (since further convolutional layers will have less parameters to tune, and the pooling layer itself has no weights) and memory consumption by extension. It will also give some tolerance to input image rotation or shift.

As with convolutional layers, the user needs to define a kernel size, stride, and a border padding. The pooling, which can be imagined as the action of taking one cell value from the matrix resulting of the projection of the pooling kernel on the input, can be accomplished by average or by maximum value. For example: in the first case, from a 2D matrix (which could have depth, therefore being 3D, but still the same applies) defined as [[2, 3], [4, 5]] the average result would be 3.5; in the second case, for that same example, the maximum would be 5. Nevertheless, figure 8 helps clarify this explanation.

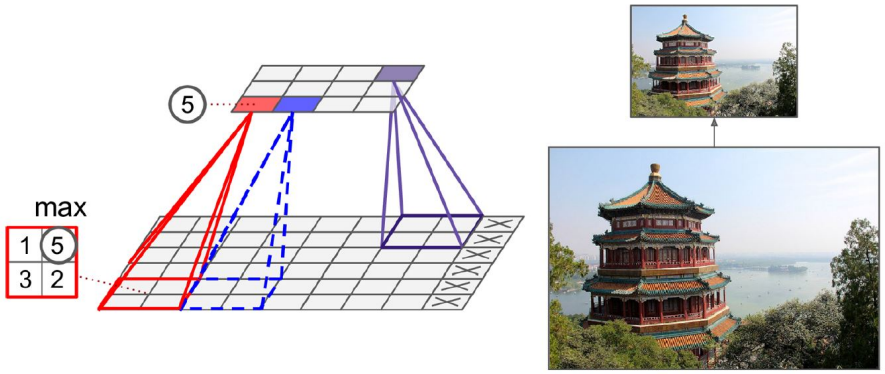

Figure 8: Image from [1]. Pooling layer in action. 2x2 kernel, stride of 2, no padding.

3 Methodology and Experiments

This section is a cronological report on how research has proceeded in this project. It details all steps that have relevance and includes all major set backs that have occurred during the work (most of them will be addressed again in the Future work section). Some cases are accompanied with code snippets. I reckon that the chronological description should help a better understanding of this work.

3.1 The web crawler

Sometimes called web spider or spiderbot, it is a piece of software to search the internet indexed in major search engines (Google, Bing) to find websites under certain *nametags* and untangle its inner structure to find the exact pages that the user requires. An example of this could be the ability of entering Amazon.com, use its search bar by inputting certain product names, and enter one by one all the results displayed and extracting their URL. It is important to note that the end objective of the crawler is to inspect the website and return the URLs that concern the products indexed in it. In fact, extracting URLs is not the only function it is conceived to do, since downloading the HTML of such pages is also within its scope. Still, that, and the specific way it is done, is left for the next step of the process.

It is important to point out that even though this step is necessary to the project (vital, in fact, since it provides the dataset on which the system will rely on), it has been skipped due to time constraints. In its substitution an already created dataset[16] publicly available in the internet of hundreds of thousands of URLs has been used. Being able to use it has made possible dedicating much more time to other parts of the work. It has to be noted that the dataset is 3 years old, and many of its entries are non-working links, but there are enough working properly ones as to have a rich dataset available. More on the web crawler is explained in the Future Work section.

3.2 The extractor

The extractor is the natural next step for the crawler. It takes the URLs of the products (or the content from which the user desires to extract information) and makes:

- A *screenshot*: Since the end of the process is a Convolutional Neural Network (CNN) that classifies items regarding their images, one needs to feed said images. To that end, a python framework called Selenium[17] is used. Selenium is designed to automate web applications for testing purposes by driving web browsers with language specific bindings, and provides enough utilities to interact in (almost) any imaginative way with web pages. In this work Selenium interacts with the engine of Google Chrome, called "chromedriver", but it also has the possibility to work with Firefox's. Every time a URL is passed to the python[18] script, a screenshot at a fixed 1280x980 resolution is taken, and stored.

- A download of the *DOM leaf text nodes*. The DOM is the Document Object Model, and according to the World Wide Web Consortium[19] is defined as "a platform and language-neutral

[16] From user Thomas Gogar, posted on GitHub at URL: https://github.com/gogartom/DOM-data-collector/tree/master/Data

[17] Selenium is a framework available for multiple programming languages, and in this case the Python one is used.

[18] Python has been chosen as the main programming language for its ease of use, popularity (which means extensive troubleshooting fixes in forums) and for the fact that the Deep Learning library used, the TensorFlow API, uses it.

[19] https://www.w3.org/DOM/

interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents. The document can be further processed and the results of that processing can be incorporated back into the presented page". If the DOM of any given page were to be displayed, it would have the shape of a tree upside down, where nested HTML tag elements form further branches. A very simple example is shown in figure 9. It is easy to mistake HTML by DOM, but they are parallel concepts. HTML is the information the server gives to the browser when it sends the request, and the DOM of the HTML is the manner in which the browser interacts with said HTML.

The leaf text nodes of the DOM are the last nodes/elements containing text in the aforementioned scheme (which, again, is a result of the HTML handed over by the requested server). The caption of Figure 9 exemplifies it. They are the content that holds the information that is being sought after. For instance, the price of a house, its square meters, or its address, to name a few. Apart from that, the bounding boxes of each element are stored. This means that the box in which the textual node of the HTML tag is located has its top, bottom, left and right pixel position downloaded (always relative to the upper left corner of the page). Like the screenshot, the content extracted from each page is conveniently stored.

```
<body>
  <div id="container_1">
    <p> Hello there</p>
  </div>
  <div id="container_2">
    <p> This is a paragraph </p>
    <p> This is another paragraph </p>
  </div>
</body>
```

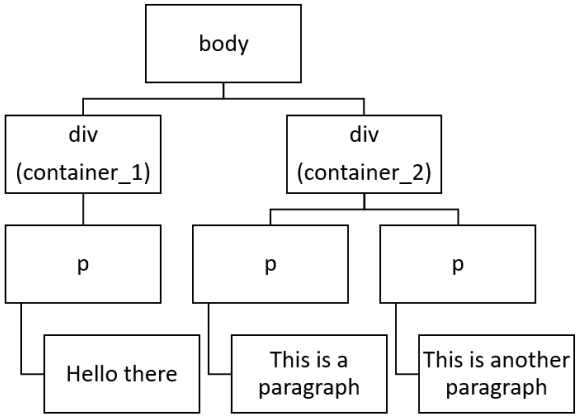

Figure 9: Illustrative example of the DOM. On the left, the HTML that a server could hand to the client/browser, and on the right the way the DOM understands it, even though by itself it is language-neutral. The leaf text nodes of this scheme would be "Hello there", "This is a paragraph" and "This is another paragraph".

The framework used with Python to implement the extractor is Selenium, as previously mentioned. It is a very useful tool, but was not designed for the exact purpose it is being used in the work. This shows both when making the screenshot and extracting the DOM leaf text nodes. These issues are addressed in the following subsections.

3.2.1 Screenshot troubleshooting

Selenium does have methods implemented to make screenshots of certain elements of the page, or even the entire page. Not only that, but it can interact with *chromedriver* in the background (called *headless mode*), or actually display the web page on screen. The obvious move would be to run the code in headless mode, since it consumes less resources, right? The problem is that most web pages get to know that the web agent is running in automated headless mode and do not display content. Running the code, therefore, forces chromedriver to display Chrome on the screen and renders the PC useless until the process is finished, which is less than ideal. But the biggest problem comes with the fact that screenshot the resolution method is bugged and does not work if it is not in headless mode. That means that there is no way to set a specific pixel height and width, let alone the image being squared. The compromise solution to this problem is by setting Windows in a resolution of 1280x1024 and making Chrome run in full-screen mode. This way, all images are 1280x980 pixels. Not squared, though, which did not seem an obstacle at the moment of coding, but proved to be a big problem when constructing the Artificial Neural Network in this project, for reasons that will be explained later.

3.2.2 DOM troubleshooting

Selenium does have methods implemented to select elements in the HTML code by their *id*, class, etc. But to the knowledge of this user, it does not have a way to call all nodes that are leaf ones (that means they do not hold any children nodes). Luckily, the framework creators seem to have foreseen that and included a method to execute JavaScript (JS). This allows to search the DOM using XML Path Language (XPath), a query language to select nodes in XML documents that also works on HTML. And JS has a built-in function to work with XPath. The code snippet is shown below:

```
var Xpath = ".//*[position()=last() and not(descendant-or-self::script
or descendant-or-self::style or descendant-or-self::noscript or
descendant-or-self::option)]"
```

The code snippet above has been tested and at this moment seems to work well. At one instance during this work the XPath search had an error that went unnoticed for weeks, which made all hard work invested in the data labelling useless, causing a great unforeseen delay in the project.

3.3 Data labelling

Neural Networks are trained by backpropagation. That means they adjust the weights in the neuronal connections based on the outcome category of the training instance. For that reason, training data must be labeled. For example, if images of dogs and cats are fed to a NN, they need to be labeled as their respective animal. In the particular case of this work, since the objective is to be able to extract product names *and* price tags, every training instance needs to have its product name's and price tag's position found by a human user. The total number of training instances a NN needs to achieve significant results tends to be very large, namely by the thousands. That means the labeling effort can not be undertaken by one single person. The easiest way to distribute the task is by creating an accessible and universal platform to which several users can connect and contribute. This has been achieved in the project by building a simple website with log in and user interface to label the images and store the results in a remote server. The user will enter an

uncomplicated page in which a screenshot from the ones extracted will be shown, and he/she will be prompted to click onto the position of the name of the product, and then the price tag, and send it as valid (position in the X and Y axes for both categories are stored when validation button is pressed). There is also the option to mark the image as inconclusive and delete it.

The web's backend (things related to the server-side) is constructed with Flask, a lightweight and easy Python framework. Python really demonstrates itself as the most all-terrain language out there. All frontend development is made using traditional methods: HTML with Bootstrap 4[20], CSS for styling[21] and JavaScript to make the web page interactive enough for its purposes. Screenshots 10 and 11 belong to the web. Data containing user input is kept in the server in a .json format, and transferred to a local PC using either FileZilla[22] or a custom Python script that uses the Paramiko framework.

[20]Recycling a template found online from GitHub user Corey Schafer in `https://github.com/CoreyMSchafer/code_snippets/tree/master/Python/Flask_Blog/02-Templates/templates`

[21]Using a CSS file from GitHub user Corey Schafer in `https://github.com/CoreyMSchafer/code_snippets/tree/master/Python/Flask_Blog/02-Templates/static`

[22]FileZilla is a File Transfer Protocol client to move files securely between machines with a user-friendly interface (which is its main strength).

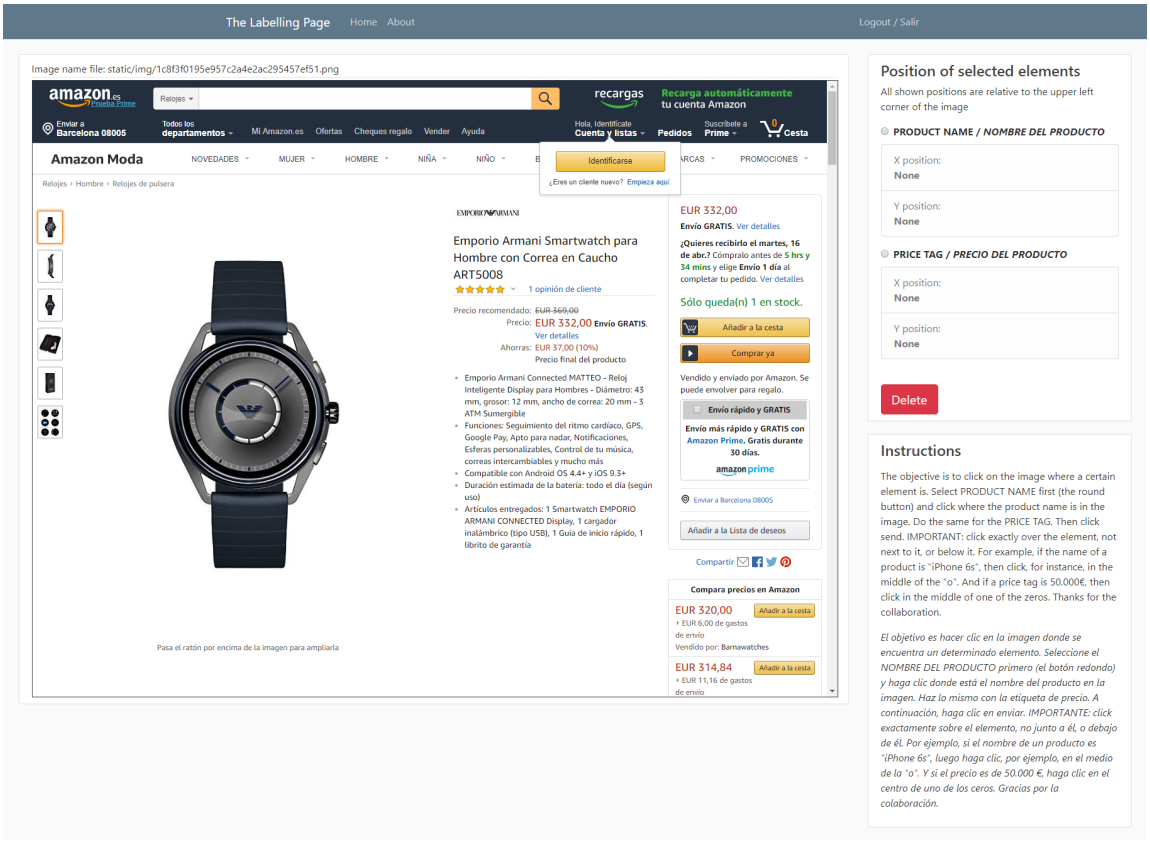

Figure 10: Image of what the user sees just after logging in. We can see the image to be labeled displayed in the center, and then the palette at the right to select the position of the items with a click (or rather delete the image if it is defective).

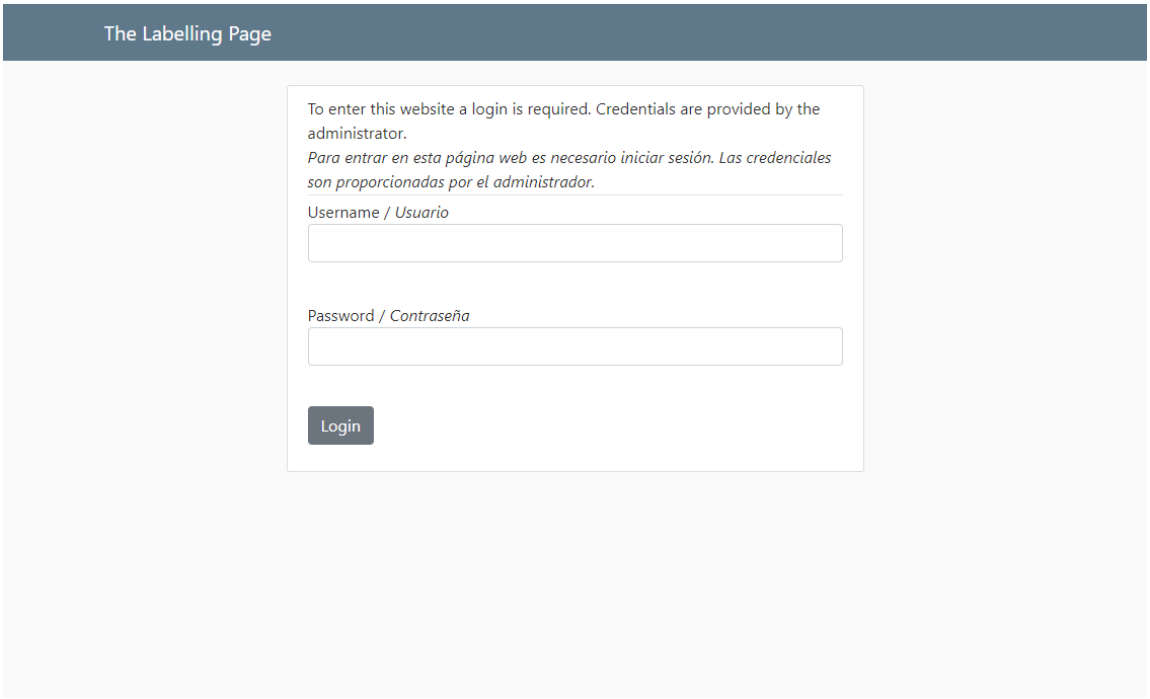

Figure 11: Image of the login page. Only users with previously given username and password can enter it. Implementing a system in which everyone can create a user would have meant creating a password recovery system, which is feasible but would have taken too much time.

3.4 Data cleaning

The extracted data are in a very raw condition. Raw in the sense that the screenshots taken are not squared (which would be desirable for reasons that will be shown later), and the DOM's leaf text nodes information are arrays of arbitrary size with possibly undesired content[23]. This requires a protocolary treatment before fed to the NN: all screenshots are to be resized at 980x980 pixels, and DOM leaf text nodes are organized in arrays of fixed size, with each text node having its position as a relative value (e.g. if the top position of an element is in pixel 100 on an image of 1000 pixels of height, its relative position is 100/1000=0.1) and its label (since labels have to be integers, 0 is assigned to "product name", 1 to "price tag" and 2 to "none of the other two").

There are also scripts designed to ensure that final dataset folders (the ones from which the NN will get its data) are consistent with each other. That means that they all have the same number of files and all are corresponded: since the NN has 3 inputs, if a file called "1c8f3f0195e957c2a4e2ac295457ef51"[24] is in a folder, it has to exist on the other two. If it does not, a message is

[23]Some websites are capable of detecting that they are being accessed by robots, and react in various ways. One would be denying access, but in some instances they will actually show the page but deliver massive false content, like megabytes of nonsense text hidden in their HTML.

[24]As the reader can see, a Universally Unique IDentifier (UUID) is used, which creates 16 bytes numbers. All files created in the project are given names with the *uuid* Python library, which takes care of doing it. Using it will ensure that there are no two different files with the same name (or the probability should at least be close to zero).

displayed and the file is deleted.

Attached to this step, a script is created to generate a parallel dataset to the one of the DOM text nodes, with the aim to provide a *spatial text encoding*. To this effect, every word inside every DOM leaf node is compared to an N-sized dictionary of words and put against a grid (a grid of arbitrary size with the same aspect ratio as the screenshot taken) in which word occurrence in space is counted. The N-sized dictionary is in fact a range of N values from 0 to N-1, since every word is hashed. The idea of the spatial text encoding is taken from [2], and from now on will also be called *textmaps*, as the author suggests in the paper. Figure 12 exemplifies it. Code below belongs to the hashing function used:

```
for idx, words in enumerate(array[:, 1]):
    words = words.split()
    top_blob = round((int(array[idx, 2])/original_H_res*H_res) / g)
    left_blob = round((int(array[idx, 3])/original_W_res*W_res) / g)
    right_blob = round((int(array[idx, 4])/original_W_res*W_res) / g)
    bottom_blob = round((int(array[idx, 5])/original_H_res*H_res) / g)
    for word in words:
        hashed_word_idx = mmh3.hash(word, 0) % N
        text_map[top_blob:bottom_blob,
                 left_blob:right_blob,
                 hashed_word_idx] += 1
```

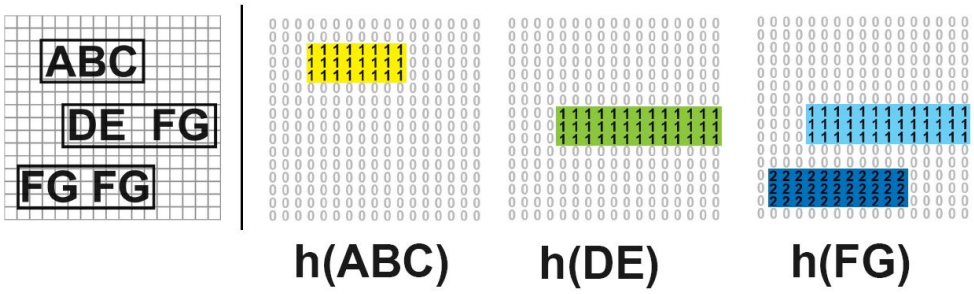

Figure 12: Image from [2]. This is the way the spatial text encoding works: every word is given a hash value from 0 to N-1, put against a grid [of zeros]. Every time one same word shows up, +1 is added to the box in which that word was found.

3.5 The Artificial Neural Network

After all data is properly formatted, it is fed to the NN to train it and hopefully be able to provide accurate predictions on the test data[25]. The following paragraphs provide an overview of its architecture, which has been based, again, on the one described in paper [2]. In fact, this same

[25]From all the data collected at the earlier step, a percentage is dedicated to training the NN (around 80%) and the remaining to validating its accuracy. It is very important to *not* feed the same data in both training and testing, since the NN may model the training data too closely (in a phenomenon known as *overfitting*, when too many training instances are fed), and would therefore give very good accuracy results that in reality are deceitful due to inability to generalize.

paper uses a second model that approximates *spatial probability distribution*. This means that, on top of the probability of each element of belonging to a certain class given by the Artificial NN, that value is multiplied by another probability that measures the chances of belonging to class X at that specific position. This spatial probability distribution model is implemented (see figure 13), but not included in the final architecture as an afterthought: the spatial probability distribution of thousands of different websites combined would render the probability map useless, since all positions in said map would have relatively high probability, hence not being useful to discern true from false. An example: in 100 instances the pricetag is located in the upper-right corner of the image; in 100 other instances in the lower-right corner; in 100 other instances in the upper-left corner; in 100 other instances in the lover-left corner; and in 100 other instances in the center. A spatial probability map from this data would be useless, since it would say that the pricetag could really be anywhere. A spatial distribution map could be useful if website-specific information extraction were to be undertaken. If the only purpose is to extract information just from Amazon.com, where prices and product names are situated more or less in the same positions, then such map would be very useful. This, though, defeats the main purpose of the work, which is to have one same architecture applicable to any commercial web.

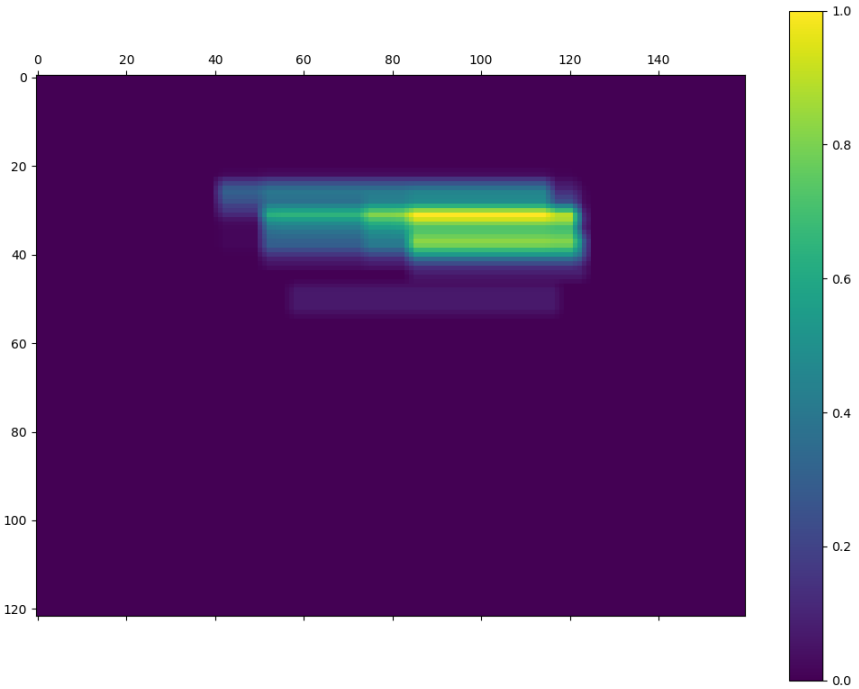

Figure 13: Plot of the probability distribution map for the class "product name" for the web Amazon.es. Legend for the colors is on the right. Positions of probability close to 1 are yellow bright, while lower probabilities are given green or even purple for null. This probability distribution map has gone through a Gaussian blur to make the edges of the probability boxes less sharp and inflexible.

Onto the Artificial NN itself: a 3 input architecture is constructed. The images (screenshots of the web pages, mentioned earlier), 980 pixels high, 980 pixels wide and 3 channels (RGB); the textmaps (also called spatial text encoding), which is decided to have 115 pixels high, 115 pixels wide and 128 channels (the hashing value N); and the array of DOM leaf text nodes, which has all candidates to be the pricetag and name of the product. In an initial test, only 10 are provided, but in a real-case scenario this number could go by the hundreds. The number of columns is of course four, which contain the spatial position of the candidate element (top, left, right and bottom positions).

The same layers as in [2] are applied, which can be seen in image 14:

1. The image is applied a convolutional layer of kernel size 11, 96 filters and stride 4. After that, a maximum-value pooling layer with kernel size 3 and stride 2. Then a convolution with

kernel size 5, 256 filters and stride 1. And yet again another convolutional layer of kernel size 3, 384 filters and stride 1.

2. The *textmaps* input is applied a single convolutional layer of kernel size 1, 48 filters and stride 1.

3. The two branches mentioned above are concatenated into a single tensor. That tensor is applied yet another convolution with kernel size 1, 48 filters and stride 1. This is where the problem of not having squared images takes place: one can not concatenate tensors that have a height and/or width mismatch. That is why images that are not squared must obligatorily be shrunk down to fulfill this requirement.

4. To the tensor resulting from the last point is then applied a custom layer called Region Of Interest Max Pooling (ROI MaxPooling for short). What this layer does is quite straightforward, but difficult to implement in code: it takes the positions of the candidates (the third input of the net mentioned earlier, the DOM leaf text nodes) whose values are relative[26] (that is, from 0 to 1) and multiplies them by the the size of the current tensor (which in the case of image 14 would be 115x115). With the new positional values, the candidate boxes (again, the candidate boxes now have positional values that are in the range of the size of the current tensor) are projected onto the tensor, and from each projection the maximum value (for every channel, of course) is taken. This layer, although conceptually easy to envision, is difficult to write, and would have not been possible without the information and code from user Jaime Sevilla in [9].

5. After last step, the tensor will have a shape in a tuple-like fashion like (number of batches[27], number of DOM candidates, number of channels). It can be imagined as a collection of vectors for every batch. To every vector for every batch a softmax layer is applied, which will output a three cells-sized vector with the probability of belonging to one of the three classes: product name, pricetag, and none. Image 15 is an example for one instance.

The NN was trained in a slow computer for testing, so no conclusions can be extracted from it (apart from the fact that it *does* compile as expected). The author has been given access to a High Performance Cluster by the project director to run the NN, but, unfortunately, there has not been enough time to do it. To make things worse, the amount of training data that needs to be manually labeled is simply too big. More on this in the conclusions.

[26] In a previous version the position values were not relative, but rather the position in pixels in the range of the input image resolution. This required interpolation and unnecessarily intricate.

[27] The NN works in batches, which means that several training instance can be fed at one same time (which will depend on computing power, but at least gives room for scalability (in the sense that a potential bottleneck is avoided when using powerful hardware).

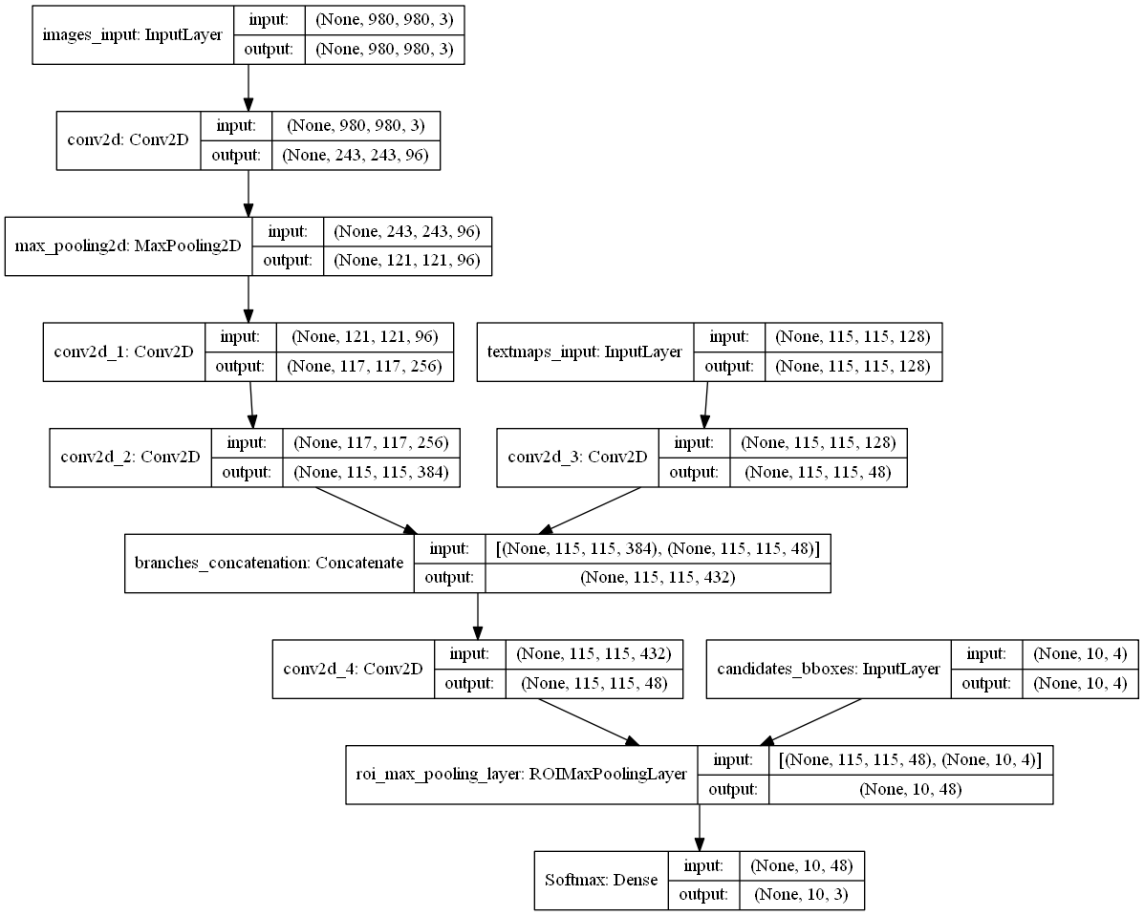

Figure 14: Plot of the NN architecture rendered by the Keras framework. Notice that "None" stands for arbitrary size. It is in fact left for the user to decide the number of instances given as input, depending on the computing power available.

Figure 15: One possible output of the NN. 10 DOM leaf text nodes were input in the NN, and those can be shown as rectangles in the image. Rectangles of color blue are for product names, red for pricetags, and black for none of the other two. As it can be seen, all have been selected as none, which is obviously wrong.

3.5.1 A final remark on the library used

There are many frameworks that provide tools to create Artificial NNs, but, currently, one of the most common ones is TensorFlow (TF), created by Google and set open-source in 2015. On top of TensorFlow runs another framework called Keras, written by François Chollet and released in the same year, which makes writing code in TF much easier. The decision on using these frameworks is simple: they are written (or at least the API) in Python, which is the language used all throughout the work, and they are the most popular frameworks in the Deep Learning community. This ensures plentiful information, blogs, Q&As, etc in the Internet, which is vital given the difficulty of some tasks.

4 Issues and Future Work

This section is very important for the project, since the initial objectives that we set for the project have not been achieved in full and further work needs to be done to reach them. The Future Work section is divided by the same subsections as Methodology and Experiments, indicating, for each, what has to be improved and what remains to be implement. At the end, all tasks mentioned are classified from highest to lowest priority, and accompanied with an estimated duration.

4.1 The web crawler and the extractor

As mentioned, the web crawler is not implemented as such, since it would have been the matter for a full project by itself and it was a task that could be skipped by resorting to a repository of URLs found in GitHub. The initial thought was to do this to gain time for other parts of the work, since the repository would provide data that were *good enough* to construct a proof of concept such as this. The problem is that the repository dates from 2016, and many of the URLs are broken or lead to products that are no longer available. The dataset does not include worldwide known product retailers like Amazon or Ebay, but rather regular e-shops from the Czech Republic, which neglects a needed variety. On the other hand, the web crawler and the extractor should be merged into one, since their roles are very similar.

There is the clear need of creating a crawler with the capability of extracting HTML code and making screenshots, but that is not a trivial task at all. A program like this should be able to enter a targeted website and search for certain items/elements in it almost like a human would. That is indeed a *very* bold statement: some pages are browsed with a search bar that accepts words, some others (like travel agencies) require clicking on certain dates, and then there are others where product entries are just shown as lists and one needs to go page per page. On top of that, as mentioned at the beginning of the report, websites generally do not like being monitored or simply browsed by a robot. They have the ability to detect they are being requested by an automated agent (or requested too many times from one single IP) and have a wide array of countermeasures: denying access, displaying a CAPTCHA, displaying false information, or even blocking the IP.

To address these issues it would be interesting to explore the Python framework Scrapy[28], designed specifically for this purpose, and scrapoxy[29] to rotate IPs and avoid being blacklisted. How to make screenshots remains unclear, but Google's framework Puppeteer[30] might well be an appropriate candidate.

4.2 Data labelling

Supposing the architecture of the NN does not change, and that a crawler/extractor is created, there will be a huge stream of incoming data. The current website created to label such data is fine for that purpose, but does not have a system to automatically create users. To be a user, a username and password must be given explicitly by the administrator, and that is less than ideal. A system should be implemented, as well as a password recovery system.

A second (and definitely cumbersome yet clever) idea could be to improve the website and make it public on Google, and implement a reward system in which users could accept "jobs" consisting of

[28] `https://docs.scrapy.org/en/latest/intro/overview.html`
[29] `https://scrapoxy.io/`
[30] https://pptr.dev/

labelling an X quantity of data instances for a Y amount of money, which would speed proceedings significantly up. To avoid boycotting or misuse, two same tasks could be given to two different people and then compared to assess if they are coherent.

4.3 Data cleaning

Ironically, the scripts designed for the cleaning of the data (always a cumbersome and time-consuming process) are not clean themselves. It is not relevant at this point providing details about the required clean-up at the moment of writing. They should be reorganised into making very specific functions, and be called by just one function, which should accept arguments for things like the location to save the data, where is the raw data located, etc. On top of that, all raw data extracted should be put in a cloud file storage system so it can be accessed from anywhere, and then pulled on demand when needed. A secondary storage system should be set in place but designed for the "clean" data (the data which has been processed in some way) that is feedable to the NN. Automated scripts could be running on the cloud system/server to ensure data consistency, freeing a PC from doing that task, which is the current case.

4.4 The Artificial Neural Network

The ANN is perhaps the piece of the project that needs most rethinking, since it is the most problematic. Said problems and future possible solutions are detailed below:

- The number of candidate DOM leaf text nodes that are input to the NN is currently fixed. It is because the NN requires to be trained with a fixed size input tensor, and it will only be capable of predicting the number of candidates it was trained with. In image 14 it is 10, for example. That is indeed a major problem, since in a real-case scenario every instance would come from a different web with different number of candidates. A possible solution could be to establish a maximum value of candidates big enough to include even the densest pages, and train the NN with it. For pages with small numbers of candidates, the ones that remain to be feedable to the NN could be filled blank.

- The architecture will not be able to predict anything more than product name, price tag and none without changing the labelling phase and retraining the NN (which is a computationally heavy task). And even in the case this is done, serious doubts remain about its ability for distinguishing certain classes from others, like distinguishing a long product name from a brief product description. One solution could be the implementation of an attention-based NN like in [10]. That would also take care of the spatial probability distribution problem.

- Related to last point: the lack of a spatial probability distribution will make it fail. As mentioned, the NN has not been fully tested, but it is unnecessary to do so in order to be able to notice that no successful results will be obtained without said spatial probability distribution map or something that alternatively does its job. An attention-based model used by the Google researchers to read street signs in [10] could and should be studied as a plausible solution.

- There is too much labelling to be done in an architecture like this. Moreover, it is not the type of simple labelling that can be found in cats *vs.* dogs problem type of NNs. Exploring an unsupervised solution should at least be considered.

- Complex product/service providers' web pages are too complex for the NN. Hotel reservation pages like Booking.com offer in one same page different prices for rooms depending on their quality. The actual NN would not be able to notice that, let alone what price corresponds to what room. This issue should be further studied, but it seems to be hard to solve, since it would mean creating a net[31] of variable output, at least in theory.
- Some retailers (like the one from the point above) do not display their prices in the upper part of the page, but rather require the user to scroll down. This is of the upmost inconvenience, since the input of the image is only 980 pixels height. A plausible solution could be to tweak the extractor/web crawler to automatically make sequential screenshots by scrolling down.
- Some information that belongs to one single category is sometimes displayed in two separate DOM leaf text nodes. This has been noticed in the way pricing is displayed in some web pages, specifically with the price number and the currency. 55$, for instance, would have an element for the 55 and another for the $ sign. An attention-based approach could theoretically solve this problem.

5 Conclusions

The proposed objectives have not been accomplished. Even though a small labelled dataset has been created and the Neural Network has been set up (with all the effort this carries behind), there has been no time to test it, nor to generate larger datasets that could help its assessment. The problem is that, even if there had been time, it can be foreseen with the current knowledge that they would be mediocre. The architecture indeed carries with it the lackings of the one it was based on. It is aparent that no "simple" computer vision solution can be applied to the information extraction problem given the multiple variations and manners of presenting information by retailers. Moreover, the are no chances of even accessing said information if the net does not have behind a robust web crawler to feed it data. The web crawler itself is a crucial component and entails an incredible difficulty and complexity, and is its own branch of research.

Still, this project has been a wonderful introduction to Deep Learning and Artificial Intelligence in general. It has explored a very little researched subject (for what it could be) and applied the latest in AI, even if it is to a basic level. The author will continue to explore and research in this area after the completion of the project, since the there is a real opportunity to make a difference in current state of the art.

[31] Short for Artificial Neural Network.

References

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

[2] T. Gogar, O. Hubacek, and J. Sedivy, "Deep Neural Networks for Web Page Information Extraction," in *12th IFIP International Conference on Artificial Intelligence Applications and Innovations (AIAI)*, ser. Artificial Intelligence Applications and Innovations, L. Iliadis and I. Maglogiannis, Eds., vol. AICT-475, Thessaloniki, Greece, Sep. 2016, pp. 154–163, part 3: Ontology-Web and Social Media AI Modeling (OWESOM).DOI: 10.1007/978-3-319-44944-9_14.

[3] D. Cai, S. Yu, J.-R. Wen, and W.-Y. Ma, "Extracting content structure for web pages based on visual representation," in *Proceedings of the 5th Asia-Pacific Web Conference on Web Technologies and Applications*, ser. APWeb'03. Berlin, Heidelberg: Springer-Verlag, 2003, pp. 406–417, DOI: doi.org/10.1007/3-540-36901-5_42.

[4] A. Arasu and H. Garcia-Molina, "Extracting structured data from web pages," in *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '03. New York, NY, USA: ACM, 2003, pp. 337–348, DOI:10.1145/872757.872799.

[5] N. Derouiche, B. Cautis, and T. Abdessalem, "Automatic extraction of structured web data with domain knowledge," in *Proceedings - International Conference on Data Engineering*, 2012, pp. 726–737, DOI: doi.org/10.1109/icde.2012.90.

[6] Z. Cai, J. Liu, L. Xu, C. Yin, and J. Wang, "A vision recognition based method for web data extraction," *Advanced Science and Technology Letters*, vol. 143, pp. 193–198, 2017, DOI: dx.doi.org/10.14257/astl.2017.143.40.

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

[8] G. Hinton and T. Tieleman. Neural networks for machine learning. Coursera. [Online]. Available: https://bit.ly/2wM1Mvy

[9] J. Sevilla. (2019) Implementing RoI pooling in tensorflow + keras. [Online]. Available: https://bit.ly/2K75g4z

[10] Z. Wojna, A. N. Gorban, D. Lee, K. Murphy, Q. Yu, Y. Li, and J. Ibarz, "Attention-based extraction of structured information from street view imagery," *CoRR*, vol. abs/1704.03549, 2017. [Online]. Available: http://arxiv.org/abs/1704.03549