Deep Neural Networks for Web Page Information Extraction

Tomas Gogar(✉), Ondrej Hubacek, and Jan Sedivy

Department of Cybernetics, Czech Technical University in Prague,
Karlovo namesti 13, Prague, Czech Republic
{gogartom,hubacon2,sedivja2}@fel.cvut.cz

Abstract. Web wrappers are systems for extracting structured information from web pages. Currently, wrappers need to be adapted to a particular website template before they can start the extraction process. In this work we present a new method, which uses convolutional neural networks to learn a wrapper that can extract information from previously unseen templates. Therefore, this wrapper does not need any site-specific initialization and is able to extract information from a single web page. We also propose a method for spatial text encoding, which allows us to encode visual and textual content of a web page into a single neural net. The first experiments with product information extraction showed very promising results and suggest that this approach can lead to a general site-independent web wrapper.

Keywords: Information extraction · Web wrappers · Convolutional neural networks

1 Introduction

The Internet is the biggest and the fastest growing source of data in today's world. Many information systems that gather structured data need to acquire information from web pages. However, HTML files are designed to be processed by web browsers and do not contain information in a structured form[1]. Therefore, systems that can extract structured information from web pages receive special attention in the research community. Such tools are usually referred to as *web wrappers*.

Although people can easily extract information from different web pages, the task of creating an automatic wrapper that can extract information from multiple websites is considered as a very complex problem. It is mainly because the semantics of elements depends on many properties such as textual content,

[1] There are efforts to include structured data in HTML, such as schema.org project, but it is still not widely used by web developers.

L. Iliadis and I. Maglogiannis (Eds.): AIAI 2016, IFIP AICT 475, pp. 154–163, 2016.
DOI: 10.1007/978-3-319-44944-9_14

visual appearance and relative positioning. Therefore, the research community is mainly focused on wrappers that need to be adapted to a particular website and then they can extract information from its web pages [3,5,8,10,16]. However, such approach brings many disadvantages, such as difficult scalability and maintenance. In this work, we show that a combination of visual and textual data in a single model can help us to create general (multi-site) wrapper. The three main contributions of this work are: (1) We propose a method of encoding data from a web rendering engine into a deep neural net - i.e. a method for spatial encoding of text. (2) On the task of product information extraction, we show that the neural net could be trained to extract information in non-trivial cases. (3) We make our dataset, source codes and final model public, in order to provide a benchmark for future work[2]. This paper is organized as follows: Sect. 2 briefly summarizes related work. In Sect. 3 we give an overview of our system, which is then described in detail in Sects. 4 and 5. In Sect. 6 we summarize our experiments and discuss achieved results. Finally, Sect. 7 summarizes this work and suggests future research.

2 Related Work

As we have mentioned in Sect. 1, current wrappers need to be adapted to a particular website before the extraction process starts. These wrappers make use of the fact that the web pages are generated from templates and thus have similar structure. The first group of wrappers uses manually labeled examples for their initialization [3,8,10]. Although these approaches achieve high accuracy, their manual nature makes them unusable in large-scale extractions. Mainly because initialization and maintenance of such wrapper for thousands of websites is not feasible. More recent works address these issues, some propose automatic wrapper maintenance [6,11], some propose methods for automatic initialization. These methods use tree-matching algorithms in order to find repeated patterns either across multiple pages [4,12,14] or within a single page [5,16] (more extensive survey of wrappers can be found in [6]). The disadvantage of these methods is their dependence on repeated patterns, which makes them unable to automatically extract information from unique document (such as invoice or product description). To the best of our knowledge, this is the first published work that addresses this problem by creating domain-specific wrapper that generalizes across previously unseen templates and does not need any site-specific initialization. Our model is inspired by deep convolutional neural networks (DCNNs) used in computer vision. Since 2012, when Krizhevsky et al. presented DCNN, which established new state-of-the-art result in image classification task [9], many other methods using convolutional nets have appeared and many of them have achieved very good results in other computer vision tasks - such as object detection [7,13] or visual segmentation [1]. These works have motivated our research, where we try to apply similar principles to a different area - Information Extraction.

[2] https://github.com/gogartom/TextMaps.

3 Architecture Overview

The semantics of elements in a semi-structured document depends on the textual content as well as on many other visual properties. Therefore, we convert data from web page DOM tree to a 2D plane and we use object detection techniques from computer vision to find DOM elements that contain the requested information. Although visual data and textual data were combined in other works on information extraction [5,16], we believe that this is the first work in this field that encodes visual and textual data into a single model. The architecture of the whole classification process is depicted in Fig. 1.

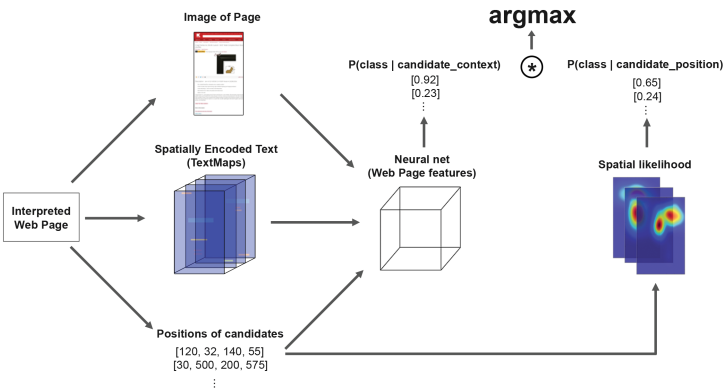

Fig. 1. System architecture overview

The following paragraphs briefly describe basic workflow of our method. In the first phase, we use the web rendering engine to fully interpret the web page and we save its screenshot and a DOM tree. From the DOM tree we are able to extract all the nodes together with their positions. Text nodes are then used to create textual input for the net (see Sect. 4.1) and leaf nodes are used as candidates for classification (candidate elements are represented by the rectangles they occupy in rendered web page). Then, we use convolutional neural net, which processes visual and textual context of candidate elements and predicts their probabilities $P(c|element\_context)$ of being in one of the predefined classes c. The target classes c are task-specific, for example the product information extraction system would classify DOM elements into four classes: *Product name*, *Main product image*, *Current final product price* and *Others*.

However, not only the close context of elements plays a role, but also absolute positioning in the web page is important. Unfortunately, convolutional networks with large inputs do not capture absolute positioning of their features and so we use training data to model spatial probability distributions $P(c|element\_position)$ (see Sect. 5), which are used to classify elements based on their absolute position. The resulting probability $P(c|element\_context)$ is

assumed to be independent of $P(c|element\_position)$ and thus we get the final prediction $P(c|element\_position, element\_context)$ by multiplication. The element with the highest resulting probability is selected as our final prediction for a given class.

4 Neural Network

Our neural network is implemented in Caffe framework and is based on a model for object detection described in [7]. In this object detection network visual features are extracted by multiple convolutional layers and a list of object proposals (represented as rectangles in input space) is then classified into particular classes. In our work, we use the same principle for classification of proposed rectangles, however, the overall architecture of our network is very different. The main difference is that our network does not process only visual but also textual data.

4.1 Spatial Text Encoding

In the field of natural language processing, texts are usually encoded in a form of vectors [2]. However, these models are not very suitable for processing of semi-structured documents, because they are designed to capture content of paragraphs and do not encode exact positions of words. In the following paragraphs, we describe our approach to text encoding, which we call *Spatial bag of words* or simply *Text Maps*.

DOM tree stores texts in *text nodes*, which can include text of various length and we can compute their bounding boxes, i.e. rectangles where all the text from the nodes is displayed (see Fig. 2). In order to encode texts effectively, we divide a page with a grid (granularity is an adjustable parameter, see Fig. 2). Using this grid we can store texts into a sparse tensor with dimensions $(N, H/g, W/g)$, where N is the size of vocabulary, H and W are page dimensions (in pixels) and g is the size of the grid cell (in pixels).

The encoding process treats each text node of a web page individually. It splits the text to individual words and for each word adds 1 to the resulting web page tensor at positions $(i, \{X\}, \{Y\})$, where $\{X\}$ and $\{Y\}$ represents indices of grid cells that are covered by text node, i is a feature index for a particular word. The feature index i is not looked up in a dictionary (as in ordinary bag-of-words settings), but as in [15], it is computed automatically by hashing a word into values between 0 and $(N - 1)$ (see example in Fig. 2). In our work we have used Murmurhash3 (32-bit signed), $N = 128$ and $g = 8$.

Note that Text Maps compress textual information a lot, we lose information about word ordering, exact positioning (because of grid size) and even exact word identity (because of index collisions). However, we get a compact representation of text, which is the same as representation of images (tensors of size $[3, H, W]$). This property is very important because it gives our model a chance to easily combine both types of information.

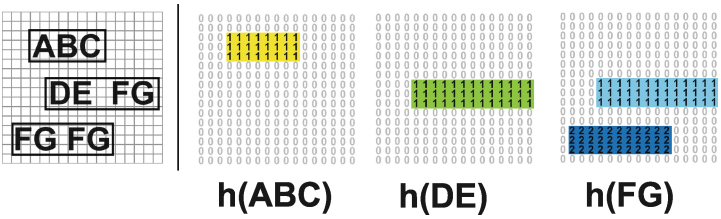

Fig. 2. Example of TextMap creation: (Left) Text nodes with words "ABC", "DE", "FG", (Right) Resulting text maps for N = 3, h() is a hashing function.

4.2 Network Architecture

This section describes architecture of our neural net which is depicted in Fig. 3. Our network has three types of inputs: *Screenshot* (we use upper crop of the page with dimensions 1280×1280, however this net can work with arbitrarily sized pages), *TextMaps* (tensor with dimensions $128 \times 160 \times 160$) and *Candidate boxes* (list of box coordinates of arbitrary length).

A screenshot is processed by three convolutional layers (the first two layers are initialized with pretrained weights from BVLC AlexNet). TextMaps are processed with one convolutional layer with kernel size 1×1 and thus its features capture various combinations of words. These two layers are then concatenated and processed by final convolutional layer. The parameters of these filters are summarized in Table 1.

The last convolutional layer has 96 channels and represents the final features of the web page (see Fig. 3). We then use ROI (Region of Interest) Pooling method [7] which allows us to take a list of bounding boxes, project them to the last layer and extract features which spatially correspond to the particular area on the page. Since DOM elements are sometimes very small, the ROI Pooling layer is set to max pool only one value from each feature map. This results in one feature vector (with 96 elements) per each candidate. These candidate vectors are then individually classified with linear classifier and softmax into final classes (4 classes in our test case). For more information on ROI Pooling, please see [7].

Table 1. Parameters of convolutional layers. C - convolutional layer, R - ReLU non-linearity, MP - max-pooling layer

Stage	Image				Text	Both
Type	C+R	MP	C+R	C+R	C+R	C+R
# Channels	96	96	256	384	48	96
Filter size	11	3	5	3	1	5
Stride	4	2	1	1	1	1

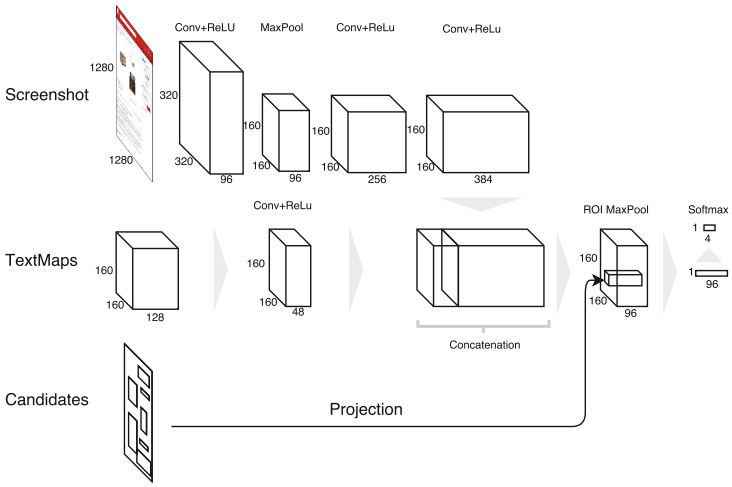

Fig. 3. Architecture of our CNN: upper part represents visual feature extraction, which is then combined with textual features (middle). Coordinates of candidate boxes are projected to final feature tensor and resulting vector is extracted using ROI MaxPooling. Finally, linear model with softmax classifies vector into predefined classes.

4.3 Training

The net was trained using stochastic gradient descent with cross-entropy loss, with learning rate $5e^{-4}$ and momentum 0.9. Mini-batch contains two images each with 100 candidate elements (3 ground truths + 97 randomly chosen others). In order to prevent overfitting, we augmented the dataset by randomly changing hue of screenshots and also by inverting page colors (with 15 % probability).

5 Spatial Probability Distribution

As mentioned in Sect. 3, we need a second model that approximates spatial probability distribution of class c. This distribution is modeled as:

$$P(c|x,y) = (f_c * g)(x,y) = \sum_{m=-M}^{M} \sum_{n=-M}^{M} f_c(x-m, y-n) \cdot g(m,n) \qquad (1)$$

where $*$ is convolution, g is 2D discrete gaussian kernel (variance was chosen experimentally). f_c is frequency matrix for class c defined as $f_c(x,y) = \frac{n_c(x,y)}{N}$, where $n_c(x,y)$ is number of samples where element of class c occupied pixel at (x,y) and N is total number of training samples. When candidate element is received (with bounding box [l,r,b,t]), its class probability given its position is computed as an average over pixels it covers:

$$P(c|elem\_position) = P(c|(l,r,b,t)) = \frac{1}{(r-l)\cdot(b-t)} \sum_{l \le x \le r} \sum_{t \le y \le b} p(c|x,y) \qquad (2)$$

6 Experiments

We test our framework on the task of product information extraction. The task is defined as follows: On a given product page, find an element that contains: *Product name*, *Main product image*, *Current final product price*. Please note, that we test the model on localization task, i.e. on every page there is exactly one ground truth element for a given class. Although this task may appear trivial on some pages, it may be actually very difficult on the others.

6.1 Data Set

Since our system requires specific data, we created a new data set, which consists of product pages from 39 online retailers from various segments. Each page in the data set consists of a screenshot and DOM tree stored in the JSON format. A ground truth label for every class is stored directly in one of the leaf elements in each DOM tree[3]. The task is to label the right leaf elements.

6.2 Baseline Models

Unfortunately, we are not aware of any model that extracts information from a single web page and that would be suitable for comparison with our work. However, we would like to explore whether our neural net was able to learn some non-trivial dependencies between elements. Therefore we compare its results with two baseline algorithms:

Spatial Distribution Baseline: The first baseline is trivial, given candidate leaf elements it selects those with highest spatial probability $P(C|candidate\_position)$.

Heuristic+Spatial Distribution Baseline: The second baseline is more complex. First, it filters candidate elements using simple textual heuristics[4]. Then, from these prefiltered elements, it selects the one with the highest spatial probability $P(C|candidate\_position)$.

6.3 Results

Using 10-fold crossvalidation we test how our model works on previously unseen sites. In Table 2 accuracy achieved on test sites after 10 thousand training iterations is presented. We can see the best results are achieved by Neural Net in combination with Spatial model. These results are comparable with web wrappers with automatic site initialization.

[3] Sources and data set: https://github.com/gogartom/TextMaps.

[4] *Price* candidates contain a dollar sign ($) and arbitrary numbers, *Image* candidates does not contain any text, *Name* candidates contain at least two unique words.

Table 2. Comparison of algorithms: mean and standard deviation of accuracy across 10 splits (in percents).

Algorithm	Image accuracy	Price accuracy	Name accuracy
NeuralNet+Spatial	**98.7 ± 1.6**	**95.3 ± 6.6**	**87.1 ± 15.0**
NeuralNet	95.9 ± 2.9	86.2 ± 9.3	78.4 ± 19.0
Baseline: Heuristic+Spatial	63.7 ± 20.1	73.6 ± 18.8	34.4 ± 20.5
Baseline: Spatial	46.5 ± 18.7	9.7 ± 14.4	12.2 ± 12.0

Table 3. Neural Net with different input data: mean and standard deviation of accuracy across 10 splits (in percents).

Neural net inputs	Image accuracy	Price accuracy	Name accuracy
Screenshot+TextMap	95.9 ± 2.9	86.2 ± 9.3	78.4 ± 19.0
Screenshot	93.5 ± 7.4	73.3 ± 19.4	73.4 ± 16.0
TextMap	41.4 ± 18.6	77.0 ± 17.9	49.4 ± 18.0

Interesting observation is the gap between Heuristic baseline and our framework in price accuracy (73.6 % vs. 95.4 %). This result suggests that Neural Net can capture price elements in non-trivial situations. Manual inspection of results confirms this hypothesis and some examples are shown in Fig. 4. We can see that our system is able to recognize *current price* elements of different sizes and distinguish them from other price tags.

Another very important observation is that localizing product name appears to be more complicated task. When examining results, we have observed that some sites divide name into two parts (manufacturer+product name) in DOM tree. Unfortunately, our net does not have capacity to distinguish between the two and we leave this issue to future work.

The last group of experiments address the influence of different input data. We have tried to train our neural net ignoring either visual or textual data and compare these results with the original net that combines both. The results are summarized in Table 3. We can see that textual data itself are not sufficient for the task, while visual data perform better. The best results are achieved by the combination of both inputs, however the difference is not significant. The biggest improvement was achieved in *current price* detection. We experimentally

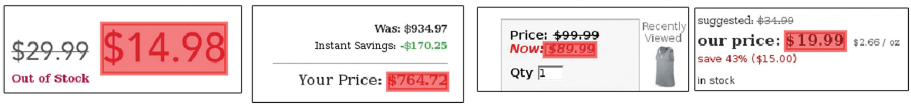

Fig. 4. Examples of *current price* detection.

verified that neural net that combines both inputs can detect spatially smaller price tags, which it is not able to recognize while using the visual features only.

7 Conclusions

In this work we have proposed a novel method for learning information extraction system that is able to generalize across previously unseen pages and therefore does not need any site-specific initialization. Since the ability of processing both - textual and visual data is crucial for this task, we have proposed a method for spatial text encoding (spatial bags-of-words). This approach allowed us to combine both types of information in one convolutional neural net. We have shown on a task of product information extraction that our model is able to generalize across web sites and can extract information in non-trivial situations (with overall accuracy 93.7 %). Achieved results are very promising and allow for immediate practical applications. However, our approach might still be improved in several ways. (i) We plan to replace the hashing function in *Text Maps* with learned representations of paragraphs. (ii) The detection algorithm may be improved in order to extract information that is stored in multiple leaf elements. (iii) And finally, simple Spatial Probability model can be replaced with more robust attention-based neural model.

Acknowledgments. This work was supported by the Grant Agency of the CTU in Prague, No. SGS16/086/OHK3/1T/13. Comp. resources provided by the CESNET LM2015042 and the CERIT Scientific Cloud LM2015085.

References

1. Badrinarayanan, V., Kendall, A., Cipolla, R.: Segnet: a deep convolutional encoder-decoder architecture for image segmentation. arXiv preprint (2015)
2. Baudiš, P., Šedivý, J.: Sentence pair scoring: towards unified framework for text comprehension. ArXiv preprints, March 2016
3. Califf, M.E., Mooney, R.J.: Bottom-up relational learning of pattern matching rules for information extraction. J. Mach. Learn. Res. **4**, 177–210 (2003)
4. Dalvi, N., Kumar, R., Soliman, M.: Automatic wrappers for large scale web extraction. Proc. VLDB Endow. **4**(4), 219–230 (2011)
5. Fan, S., Wang, X., Dong, Y.: Web data extraction based on visual information and partial tree alignment. In: 2014 11th Web Information System and Application Conference (WISA), September 2014, pp. 18–23 (2014)
6. Ferrara, E., Meo, P.D., Fiumara, G., Baumgartner, R.: Web data extraction, applications and techniques: a survey. Knowl. Based Syst. **70**, 301–323 (2014)
7. Girshick, R.: Fast R-CNN. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 1440–1448 (2015)
8. Hsu, C.N., Dung, M.T.: Generating finite-state transducers for semi-structured data extraction from the web. Inf. Syst. **23**, 521–538 (1998)
9. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems (2012)

10. Kushmerick, N.: Wrapper induction: efficiency and expressiveness. Artif. Intell. **118**(1–2), 15–68 (2000)
11. Ortona, S., Orsi, G., Buoncristiano, M., Furche, T.: Wadar: joint wrapper and data repair. Proc. VLDB Endow. **8**(12), 1996–1999 (2015)
12. Qiu, D., Barbosa, L., Dong, X.L., Shen, Y., Srivastava, D.: Dexter: large-scale discovery and extraction of product specifications on the web. VLDB Endow. (2015)
13. Sermanet, P., Eigen, D., Zhang, X., Mathieu, M., Fergus, R., LeCun, Y.: Overfeat: integrated recognition, localization and detection using convolutional networks. ArXiv preprint (2013)
14. Brambilla, M., Tokuda, T., Tolksdorf, R. (eds.): ICWE 2012. LNCS, vol. 7387. Springer, Heidelberg (2012)
15. Weinberger, K., Dasgupta, A., Langford, J., Smola, A., Attenberg, J.: Feature hashing for large scale multitask learning. In: Proceedings of the 26th Annual International Conference on Machine Learning, ICML 2009 (2009)
16. Zhai, Y., Liu, B.: Automatic wrapper generation using tree matching and partial tree alignment. In: Proceedings of the National Conference on Artificial Intelligence (2006)