Genre-Oriented Web Content Extraction with Deep Convolutional Neural Networks and Statistical Methods

Bao-Dai Nguyen-Hoang
Knorex Vietnam Co., Ltd.
46 Bach Dang, Tan Binh,
Ho Chi Minh city, Vietnam
dai_nguyen@knorex.com

Bao-Tran Pham-Hong
Knorex Vietnam Co., Ltd
46 Bach Dang, Tan Binh
Ho Chi Minh city, Vietnam
tran_pham@knorex.com

Yiping Jin
Knorex Pte. Ltd.
2 Science Park Drive,
Singapore 118222
jinyiping@knorex.com

Phu T. V. Le
Knorex Pte. Ltd.
2 Science Park Drive,
Singapore 118222
le.phu@knorex.com

Abstract

Extracting clean textual content from the Web is the first and an essential step to resolve most of down-stream natural language processing tasks. Previous works in web content extraction focus mainly on web pages with a single main block of textual content, such as news articles and blog posts. They employ techniques that rely largely on the HTML structure of the page to extract the main content.

Little attention has been paid to recognizing different genres of web pages, which can have a tremendous impact on the accuracy of the content extraction algorithms. In this paper, we propose Genre-Oriented Content Extraction (GOCE), a novel content extraction framework consisting of two steps: web genre classification and content extraction algorithms based on the detected genre. GOCE first employs state-of-the-art convolutional neural network models to classify the genre of the web page given a rendered image of the page. Then it applies content extraction algorithm utilizing the genre information for a more robust and accurate textual content extraction. Experiments show that GOCE achieves promising results in both web genre classification and the end-to-end content extraction task. Furthermore, GOCE greatly improves from a well-cited competitive baseline. We will also publish our dataset which contains annotations of both the web genre and the gold-standard textual content.

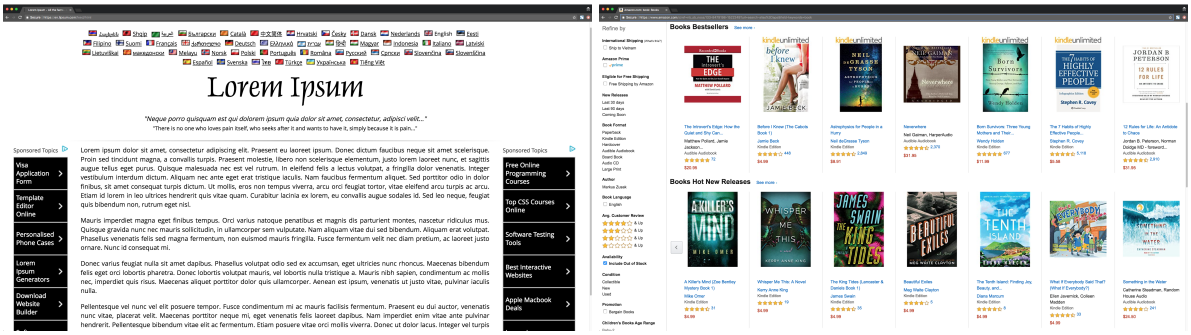
1 Introduction

Web content extraction is the task of pulling the main textual content out of web pages while removing noise such as clutters (for e.g. HTML tags, scripts, etc.) and boilerplate contents (for e.g. navigation bar, headers, footers, etc.) (Gupta et al., 2003). Humans can easily identify the main content based on the layout and visuals of web pages. However, it is not trivial for a computer program to accurately detect the meaningful content and skip the noise due to the complex and dynamic content layout of web pages.

Content extraction algorithms promise multiple applications by extracting only the important and relevant content from web pages. Firstly, the extracted content is more readable for humans, which is an important criteria for third-party data consumers, such as an application that subscribes to the content of a website. Secondly, by removing the noisy textual content such as advertisements, headers and footers from the web page, it helps to improve the performance of down-stream natural language processing tasks, such as text classification (Jena and Kamila, 2013) and text summarisation (Joseph and Jeba, 2016).

Kohlschütter et al. (2010) introduced a content extraction algorithm that uses heuristic rules to detect and eliminate boilerplate content. Boilerplate content describes sections of code that have to be reused repeatedly in HTML documents, which does

32nd Pacific Asia Conference on Language, Information and Computation
Hong Kong, 1-3 December 2018



(a) Sample of article page

(b) Sample of list-view page

Figure 1: Two examples of different web genres

not contribute to the main content. The method detects and removes short text spans (likely appearing in navigation bars), and only keep long text spans (likely the main content). The authors defined density-specific rules to classify atomic text blocks into boilerplate or content category. They showed that boilerplate detection using shallow text features is an effective approach, with the assumption that main content is likely followed by main content and clutter is likely followed by clutter. Their method outperformed previous methods on multi-domain corpora consisting of news articles and blogs pages.

However, most existing works on website content extraction (Kohlschütter et al., 2010; Hssina et al., 2013) attempt to pull the main content based only on HTML document using heuristic rules, which might result in three main shortcomings. Firstly, it is prone to wrongly removing short but important text spans due to hard-coded rules. Secondly, when the main content of the page are not adjacent to each other (e.g. separated by a block quote or are located in different sections), the algorithm often fails to extract the whole content, but extracts the most prominent chunk instead. Thirdly, the algorithm is engineered for web pages with a single main text span. it may not work for list-view page, where important content spreads across numerous tags (e.g. tag). This genre of pages contributes to a large proportion of content in the Internet, such as product listing, forums and review pages.

Figure 1 depicts two different genres of web pages: *article* and *list-view* page. The article page has the main content located in the center of the page. The page may optionally contain extraneous content, such as navigation bars and comments. The list-view page has the main content spread across different parts of the page. The text spans are of various length but in general are much shorter than the text spans in article pages. We hypothesise that the overall content extraction accuracy will be improved if we apply different extraction algorithm for different genres of web pages.

This paper presents Genre-Oriented Content Extraction (GOCE), a novel 2-stage system that automatically extracts the main content from different genres of web pages. GOCE first uses convolutional neural network (CNN) (Krizhevsky et al., 2012) models to detect the genre of web page based on the rendered image of the web page. It then applies extraction algorithm guided by the detected genre to retain only the main relevant content. Our experiments showed that GOCE achieves promising results in both web genre classification and the end-to-end content extraction task. Furthermore, GOCE greatly improves from a well-cited competitive baseline.

Overall, the contributions of this paper are:

- Introduce an image classifier to identify genres of the web pages using convolutional neural networks.
- Improve the article extraction method and outperform the boilerplate removal pipeline (Kohlschütter et al., 2010) by a large margin.
- Propose a novel method to extract main content

from list-view web pages.

- Provide a complete evaluation framework and a manually labelled dataset to evaluate end-to-end content extraction algorithms for different genres of web pages.

The rest of this paper is organized as follows. Section 2 introduces the related works. Section 3 describes the web genre detection model and our approach to extract main content from article pages and list-view pages. Experimental results are presented in Section 4. Finally, we conclude the paper and point out directions for future works.

2 Related work

In this section, we introduce three areas of research which are related to this work: web content extraction, web genre classification and image classification.

2.1 Content Extraction

Many previous works in content extraction focus on applying manually crafted or automatically mined rules on HTML documents to remove the noise. Sandip et al. (2005) proposed a method to identify the primary informative content of a Web page. After that, they detect primary informative content block and then remove all the irrelevant blocks from HTML document. Jinbeom Kang et al. (2010) introduced repetition-based page segmentation to recognize repetitive tag patterns in the DOM tree structure of a page. Badr Hssina et al. (2013) used hand-crafted rules to remove noise present in the HTML document. Their hand-crafted rules manipulate string not only to remove noise but also to extract common patterns where the main content often locates. Kohlschütter et al. (2010) proposed an approach for boilerplate detection using shallow text features, which can effectively remove irrelevant content such as navigational boilerplate text.

Apart from HTML-based methods, another direction of research is to exploit the visual information besides the HTML document. Cai et al. (2003) introduced Vision-based Page Segmentation algorithm which extracts web content by simulating how a user understands web layout structure based on his visual perception. This approach is independent to underlying documentation representation and makes full use of page layout features in HTML document. Wei Liu et al. (2010) explored the visual regularity of the data records and data items on web pages, and came up with ViDe, a vision-based approach for web data extraction. Nethra and Anitha (2014) proposed a content extraction algorithm integrating textual and visual scores to extract informative content from web. Textual and visual scores are calculated based on each node in the DOM tree converted from HTML document. Their algorithm derives hybrid density from textual and visual scores and uses it to extract the informative content from web pages. Lavanya and Dhanalakshmi (2013) provided structured and comprehensive vision-based features of research efforts made in the field of web content extraction.

2.2 Web genre classification

Eissen and Stein (2004) presented results from the construction of a web genre classifier using discriminant analysis, neural network learning, and support vector machines. Boese and Howe (2005) examined the effects of page evolution on genre classification of web pages. Santini (2007) analyzed the web genre classification problems in terms of two broad phenomena: genre hybridism and individualization. This not only helps pinpoint the range of flexibility, but also accounts for multi-genre variation of the individual web page. Dong et al. (2008) described a set of experiments to examine the effect of various attributes of web page on web genre classification task. Their results indicated that fewer features produce better precision but more features produce better recall, and attributes in combinations will always outperform single attribute.

2.3 Image Classification

Deep Convolutional Neural Networks (Krizhevsky et al., 2012; LeCun et al., 1989) have led to a series of breakthroughs for image classification (Sermanet et al., 2013; Zeiler and Fergus, 2014). Deep networks can automatically derive features from raw image pixels and generate different levels of representation (Zeiler and Fergus, 2014). The levels of features can be enriched by stacking more layers to the network (He et al., 2016a). Krizhevsky et al. (2012) proposed a deep CNN to classify

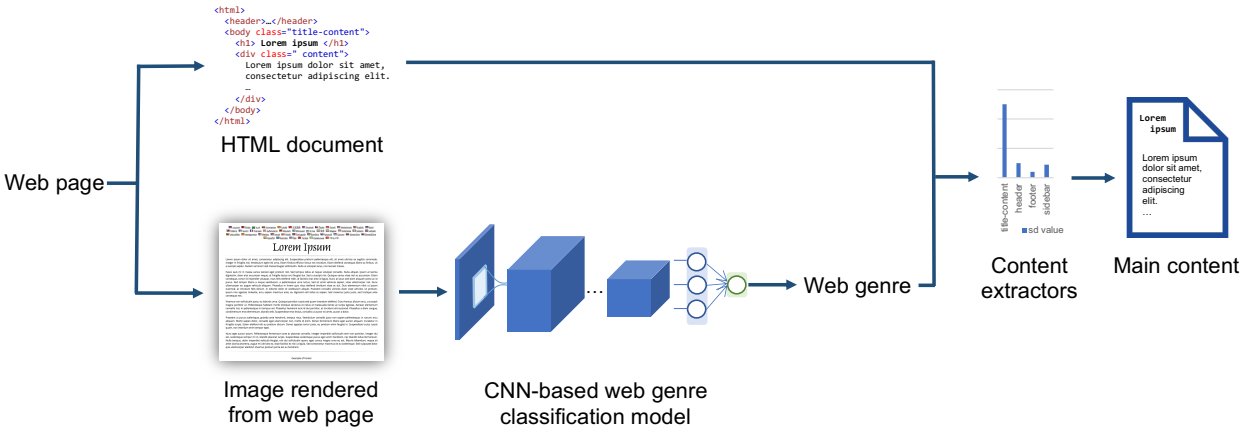


Figure 2: Genre-oriented content extraction pipeline

1.2 million high-resolution images in the ImageNet LSVRC-2010 contest. Their network achieved top-1 and top-5 error rates of 37.5% and 17.0%, which is considerably better than the previous state-of-the-art using six sparse-coding models trained on different features (A. Berg and Fei-Fei., 2010). Since then, CNNs has become the dominant model family for image classification.

Szegedy et al. (2015) proposed a deep convolutional neural network architecture code-named Inception, with top-1 and top-5 error rates of 30.2% and 10.4% on ImageNet LSVRC-2012 contest. He et al. (2016a) firstly proposed ResNet, another convolutional-based network, which has layers as learning residual functions with reference to the layer inputs, instead of learning unreferenced functions. Szegedy et al. (2016b) introduced ResNet V2 using identity mappings as the skip connections and after-addition activation. Their report showed that with top-1 and top-5 error rates of 23.0% and 6.3% on ImageNet LSVRC-2012 and 4.62% error on CIFAR-10 dataset, ResNet V2 improved accuracy on image classification task compared with previous models.

3 Proposed method

3.1 GOCE - Genre-Oriented Content Extraction pipeline

To overcome the challenges for content extraction due to different genres of web pages, we propose Genre-oriented content extraction (GOCE), which applies extraction algorithm guided by the detected genre.

Figure 2 shows the high-level architecture of GOCE. Firstly, we download the HTML document and render the web page as an image using a headless-browser library. Secondly, we feed the image to a CNN-based image classification model to classify whether the web page is an *article* page or a *list-view* page. Based on the classified page genre, GOCE applies the appropriate content extraction algorithm.

3.2 Web Page Genre Classification

We train deep CNN models to classify the genre of the web page based on the rendered image. Instead of training from scratch, we fine-tune a pre-trained model which was trained on the ImageNet dataset. The pre-trained model can capture task-independent representations in the internal layers, which can be adapted to another target task without retraining (Yosinski et al., 2014). We follow the common practice of model fine-tuning: we freeze the weights of all internal layers and retrain only the final logits layer which predicts the two classes: *article* page and *list-view* page.

3.3 Content Extraction Algorithms

GOCE applies the appropriate content extraction algorithm based on the classified genre of the page. In this section, we introduce two novel content extraction algorithms designed for the two genres of web pages: *article* page and *list-view* page.

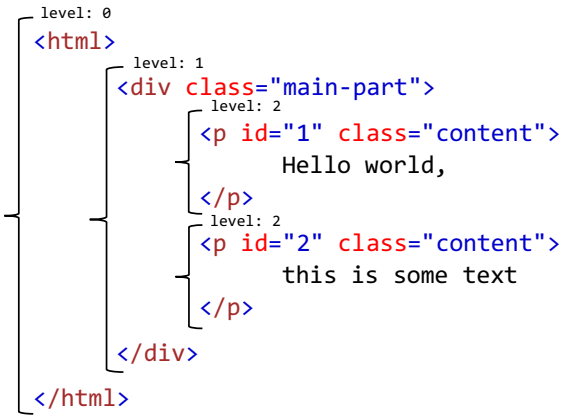


Figure 3: Example of an HTML document

3.3.1 Terminologies

Figure 3 shows a toy example of an HTML page. We treat each whole tag in HTML document as an *element* (parts covered by curly brackets). The *element* might have one or more child elements and associated text. For instance, the textual content of the <p> element with `id="1"` is "`Hello world,`", and the <p> element with `id="2"` is "`this is some text`". Meanwhile, tag has two <p> tags as child elements, which means the text attribute of the tag is "`Hello world,\n this is some text`".

Each element can have zero or more class names. For example, element has the class "`main-part`". Last but not least, we also concern about the depth level of each class name in the HTML document. The depth level of each class name is determined by the position of its element in the DOM tree. The <html> tag represents the root of an HTML document, which has depth level 0. From that, we plus 1 to obtain the depth level of the one-step deeper element. For example, element has a depth level 1, since it is the direct child node of <html> element.

3.3.2 Article algorithm

Algorithm 1 summarises our recursive article extraction algorithm for article pages. We take the first *element* of the website and check the number of its child elements. We will return the text in the *element* tag if no child element is found, or traverse to the child recursively if only one child is found.

Firstly, we calculate the standard deviation (SD) of the length of all child elements, when the number of child elements is greater than or equal to 2. SD value represents the variation of a text length among child elements. A high SD value indicates that the child elements have very different text length, which suggests that the main content should locate only in elements with large amount of text. On the other hand, a low SD indicates that the text length of the child elements are roughly equal. Therefore, the main content is distributed more evenly among the elements. Secondly, we compare the difference in length of the two longest text spans among the child elements with the SD value. If the difference is greater than the SD value, we extract the textual content of only the child node with the longest text. Otherwise, we extract the whole text of the parent node.

Algorithm 1 Article extractor

```
1:  function ARTICLEEXTRACT(element)
2:      childEle ← element.allChild
3:      if childEle.length == 0 then
4:          return element.text
5:      if childEle.length == 1 then
6:          return articleExtract(childEle[0])
7:      m1, m2 ← getTwoMax(childEle)
8:      lm1 ← wordCount(m1.text)
9:      lm2 ← wordCount(m2.text)
10:     stdDev ← calcStdDev(childEle)
11:     if (lm1 − lm2) > stdDev then
12:         return articleExtract(m1)
13:     else
14:         return element.text
```

3.3.3 List-view algorithm

Algorithm 2 summarises the list-view extraction algorithm. The algorithm identifies the main content by ranking all the class names with their depth levels, and choose only one which represents the elements containing the main content.

We define the main function in line 13, which takes the first *element* of the HTML document and the n value as input. The n value denotes the number of candidate classes to be considered. In line 14, we iterate all the elements in the HTML document recursively to retrieve the number of occurrences and corresponding text of each class. These information are stored in two global dictionaries, *id-*

Algorithm 2 List-view extractor

```
1:  global variables
2:      idCount - occurrences of IDs
3:      idText - text of IDs
4:  end global variables
5:  function UPDATEINFO(element, level)
6:      id ← combine(level, element.className)
7:      idCount[id] ← idCount[id] + 1
8:      idText[id] ← idText[id] + element.text
9:  function RETRIEVEELE(element, level)
10:     updateInfo(element, level)
11:     for child in element.child do
12:         updateInfo(child, level + 1)
13: function LISTVIEWEXTRACTOR(element, n)
14:     retrieveEle(element, 0)
15:     idR ← calcRScores(idCount, idText)
16:     topNId ← getTopNId(idR, n)
17:     predictedId ← getMaxATL(topNId)
18:     return predictedId.text
```

Count and *idText*. We assume that elements with the same class name and depth level contain related content. Meanwhile, elements with different depth level may not represent related content even if they share the same class name. Therefore, we concatenate the class name and depth level to form the key ID (which is the *id* variable defined at line 6 in Algorithm 2).

$$R = \frac{2OL}{O + L} \tag{1}$$

$$ATL = \frac{L}{O} \tag{2}$$

After parsing the elements and populating the dictionaries, we compute the ranking score R based on the occurrences O of the ID and the length L of the text from the corresponding ID, as shown in equation 1. Intuitively, the main content of a list-view page is located in elements which not only have a significant amount of text, but their attached class name also appears often in elements with the same depth level. We rank the IDs based on their R scores to have the list of top *n* IDs. After that, we compute the average text length ATL value for each ID in the top-*n* list using equation 2.

Finally, the main content is extracted from all elements occurring with the ID which has the maximum ATL value. We choose the top $n = 15$ candidate classes to evaluate our GOCE's list-view algorithm.

Method	Acc	P/R/F_1
SVM	87.8	87.8/87.8/87.8
Inception V3	83.0	82.9/86.7/84.8
Inception V4	79.3	80.0/82.6/81.4
VGG 16	91.4	88.7/**96.9**/92.6
VGG 19	86.0	86.9/88.2/87.5
ResNet V2 101	**94.8**	**95.8**/94.5/**95.2**
ResNet V2 152	92.5	90.2/96.8/93.4
Inception ResNet V2	74.5	74.8/80.3/77.4

Table 1: Accuracy and Precision/Recall/F1 scores (%) on web genre classification task

4 Experiments

4.1 Datasets

We manually annotated a dataset to evaluate our model performance. Our dataset contains randomly sampled web pages from the Internet. We annotated the genre of the web pages with one label from *S = {article, list-view}*.

The dataset includes 2,372 article pages and 3,350 list-view pages. We rendered images from this dataset using PhantomJS library[1]. We used this set of images to train and evaluate our web genre classifier. Out of the dataset, we randomly picked 100 article documents and 100 list-view documents, then manually curated the main content to build a gold evaluation dataset for content extraction algorithms.

4.2 Evaluations on Web Genre Classifiers

In this section, we present evaluations on web genre classification. To train the web genre classifier, we fine-tuned pre-trained models with TensorFlow-Slim toolkit[2], using the gold image dataset with a random 90%/10% train/test split. We benchmarked some popular CNN models, including Inception V3 (Szegedy et al., 2016), Inception V4 (Szegedy et al., 2017), VGG 16, VGG 19 (Simonyan and Zisserman, 2014), 101-layer ResNet V2, 152-layer ResNet

[1] https://github.com/ariya/phantomjs

[2] https://github.com/tensorflow/models/tree/master/research/slim

Data	Metrics	Default	Boilerpipe	Article	List-view	GOCE	Oracle
Article pages only	*Cosine similarity*	80.72	92.78	**94.70**	66.38	92.30	-/-/-
	Precision	39.07	80.57	**81.31**	53.93	79.81	
	Recall	**98.21**	91.53	95.22	57.28	92.80	
	F_1	55.90	85.70	**87.72**	55.55	85.82	
List-view pages only	*Cosine similarity*	76.83	48.76	72.53	78.16	**80.07**	-/-/-
	Precision	46.31	56.84	63.54	71.21	**74.53**	
	Recall	**86.95**	22.11	64.27	72.91	72.23	
	F_1	60.43	31.84	63.90	72.05	**73.36**	
Full dataset	*Cosine similarity*	78.77	70.77	83.62	72.27	**86.16**	86.43
	Precision	42.69	68.70	72.43	62.57	**77.04**	76.26
	Recall	**92.58**	56.82	79.75	65.10	82.62	84.06
	F_1	58.43	62.20	75.91	63.81	**79.74**	79.97

Table 2: Comparison between using default mode, boilerpipe, article algorithm only, list-view algorithm only, full pipeline of GOCE, and oracle mode (%) on two different data subsets and combined dataset

V2 (He et al., 2016b), and combined Inception-ResNet V2 model (Szegedy et al., 2017). We stopped the fine-tuning process until the model converges.

We also compared deep CNN models with a SVM baseline implemented using the Weka toolkit (Hall et al., 2009). Based on an empirical analysis from article and list-viewed pages, we came up with 11 features derived from the HTML document to train the SVM model. The complete list of features is as follows:

1. Text length of the whole document
2. Number of clusters containing elements having the same class name, tag name, number of child nodes, child tag name, and child class name with others in their cluster
3. Occurrences of elements having the same class name, tag name, number of child nodes, child tag name, and child class name with other one or more elements
4. Number of image URLs
5. Number of periods (.)
6. Number of commas (,)
7. Number of semicolons (;)
8. Number of colons (:)
9. Number of question marks (?)
10. Number of exclamation marks (!)
11. Number of digits (from 0 to 9)

Table 1 shows the performance of the SVM baseline and various CNN-based networks on our dataset. We can see that the 101-layer ResNet V2 achieved the highest accuracy and F_1 score. ResNet follows a principled approach to add shortcut connections every two layers to a VGG-style network (Simonyan and Zisserman, 2014). It not only facilitates training process but also helps model achieve both lower training and test errors. The result agrees with studies in previous work, which evaluated using the ImageNet dataset (He et al., 2016b).

4.3 End-to-End Evaluations on Genre-Oriented Content Extraction

Following Bär et al. (2015), we used two evaluation metrics to evaluate the content extraction performance. The first is to compute precision, recall, and F_1 score based on Longest Common Subsequence (LCS) (Hunt and Szymanski, 1977). We denoted the manually curated content as *gold sequence*, and the content which content extractor outputs as *extracted sequence*, and the LCS between these two sequences as *gold* $\cap$ *extracted sequence*. Precision p and recall r can be computed respectively using equation 3 and 4. The second evaluation metric is the cosine distance between the extracted content and gold content when represented as bag-of-word vectors w_i and w_j (as shown in equation 5).

$$p = \frac{\text{length}(\textit{gold} \cap \textit{extracted sequence})}{\text{length}(\textit{extracted sequence})} \quad (3)$$

$$r = \frac{\text{length}(\textit{gold} \cap \textit{extracted sequence})}{\text{length}(\textit{gold sequence})} \quad (4)$$

$$\cos\theta = \frac{w_i.w_j}{||w_i||||w_j||} \quad (5)$$

To evaluate our approach in the end-to-end manner, we implemented the full GOCE pipeline using 101-layer ResNet V2 as the web genre classifier. We compared GOCE with two baselines, including Boilerpipe method (Kohlschütter et al., 2010) and default mode, which extracts all the content of the web page. Furthermore, we also reported the results using only one algorithm of GOCE: article mode or list-view mode. This helps conduct a fair and comprehensive evaluation, since Boilerpipe was designed to extract content from article web pages only. To investigate whether the error in genre classification plays an important role in the end-to-end content extraction accuracy, we also showed the performance of the oracle mode, which uses the oracle web page genre to choose the right content extraction algorithm. We evaluated these end-to-end content extraction approaches not only on full dataset but also on two sub-datasets, which are article web pages only and list-view web pages only.

The results of the different methods on content extraction task are summarised in Table 2. On the article dataset, we can see that Boilerpipe method achieved a better performance compared with default mode in terms of cosine similarity and F_1 score. However, Boilerpipe method performed poorly for list-view dataset with 31.84% on F_1 score, which suggests the necessity of another algorithm to extract contents from list-view pages. Besides, article mode outperformed Boilerpipe across all datasets. In terms of cosine similarity and F_1 score, article mode improved the performance of Boilerpipe by 1.92% and 2.02% on the article dataset, and by 12.85% and 13.71% on the full dataset. Oracle result on the sub-datasets is not shown since they are identical with result obtained from corresponding mode of that web genre.

Excluding GOCE, article and list-view mode performed best on their corresponding corpus, which shows that knowing the genre helps to improve the performance of content extraction. Including GOCE, it is noteworthy that the performance of GOCE is slightly better than list-view mode on the list-view dataset, since there are some list-view pages wrapping the whole main content in one element without separating into smaller child nodes. GOCE classified these pages as article and had better result extracting with article mode. The full GOCE pipeline achieved the best cosine similarity and F_1 score on full dataset. In terms of F_1 score, GOCE yielded a further improvement of 17.54% compared with Boilerpipe, and 3.83% and 15.93% compared with article mode and list-view mode, respectively. This validates that our 2-stage pipeline using the information of the web genre significantly improves the performance of content extraction systems. However, there is still a gap between GOCE and oracle mode due to incorrect web genre classifications, which decreases F_1 score by 0.23%. It motivates our future work to improve the individual extraction algorithms prior to the web genre classification model.

5 Conclusion and future work

In this paper, we proposed Genre-Oriented Content Extraction (GOCE) on HTML documents by using a 2-stage pipeline: we firstly identify the genre of the web page and extract the main content based on the detected genre.

We fine-tuned a CNN-based model, especially ResNet architecture, to predict web genre based on the rendered image of the web page. The classifier achieves a promising F_1 score of 95.20%. We also proposed recursive statistical methods to extract the main content from article pages and list-view pages. The full GOCE model outperformed the competitive baseline by 17.54% in terms of F_1 score.

In future work, we plan to upgrade the GOCE pipeline to a multi-task model that can predict the genre of web page and its main content jointly. We are also working on improving individual extraction algorithms and extending web genre classifier to other genres of web pages, such as home pages and image/video pages.

References

J. Deng A. Berg and L. Fei-Fei. 2010. Large scale visual recognition challenge 2010.

Daniel Bär, Torsten Zesch, and Iryna Gurevych. 2015. Composing measures for computing text similarity.

Elizabeth Sugar Boese and Adele E Howe. 2005. Effects of web document evolution on genre classification. In *Proceedings of the 14th ACM international conference on Information and knowledge management*, pages 632–639. ACM.

Deng Cai, Shipeng Yu, Ji-Rong Wen, and Wei-Ying Ma. 2003. Extracting content structure for web pages based on visual representation. In *Asia-Pacific Web Conference*, pages 406–417. Springer.

Sandip Debnath, Prasenjit Mitra, Nirmal Pal, and C Lee Giles. 2005. Automatic identification of informative sections of web pages. *IEEE transactions on knowledge and data engineering*, 17(9):1233–1246.

Lei Dong, Carolyn Watters, Jack Duffy, and Michael Shepherd. 2008. An examination of genre attributes for web page classification. In *hicss*, page 133. IEEE.

Suhit Gupta, Gail Kaiser, David Neistadt, and Peter Grimm. 2003. Dom-based content extraction of html documents. In *Proceedings of the 12th International Conference on World Wide Web*, WWW '03, pages 207–214, New York, NY, USA. ACM.

Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. 2009. The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016a. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016b. Identity mappings in deep residual networks. In *European Conference on Computer Vision*, pages 630–645. Springer.

Badr Hssina, Abdelkarim Merbouha, Hanane Ezzikouri, Mohammed Erritali, and Belaid Bouikhalene. 2013. An implementation of web content extraction using mining techniques. *Journal of Theoretical & Applied Information Technology*, 58(3).

James W Hunt and Thomas G Szymanski. 1977. A fast algorithm for computing longest common subsequences. *Communications of the ACM*, 20(5):350–353.

Lambodar Jena and Narendra Kumar Kamila. 2013. Data extraction and web page categorization using text mining. *IJAIEM, ISSN*, pages 2319–4847.

Jincymol Joseph and JR Jeba. 2016. Survey on web content extraction. *International Journal of Applied Engineering Research*, 11(7):5338–5341.

Jinbeom Kang, Jaeyoung Yang, and Joongmin Choi. 2010. Repetition-based web page segmentation by detecting tag patterns for small-screen devices. *IEEE Transactions on Consumer Electronics*, 56(2).

Christian Kohlschütter, Peter Fankhauser, and Wolfgang Nejdl. 2010. Boilerplate detection using shallow text features. In *Proceedings of the Third ACM International Conference on Web Search and Data Mining*, WSDM '10, pages 441–450, New York, NY, USA. ACM.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

M Lavanya and M Dhanalakshmi. 2013. Various approaches of vision-based deep web data extraction (vdwde) and applications. 4(1).

Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. 1989. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551.

Wei Liu, Xiaofeng Meng, and Weiyi Meng. 2010. Vide: A vision-based approach for deep web data extraction. *IEEE Transactions on Knowledge and Data Engineering*, 22(3):447–460.

K Nethra and J Anitha. 2014. Web content extraction by integrating textual and visual importance of web pages. *International Journal of Computer Applications*, 91(3).

Marina Santini. 2007. Characterizing genres of web pages: Genre hybridism and individualization. In *System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on*, pages 71–71. IEEE.

Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, and Yann LeCun. 2013. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*.

Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich, et al. 2015. Going deeper with convolutions. Cvpr.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. 2016. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2818–2826.

Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. 2017. Inception-v4, inception-

resnet and the impact of residual connections on learning. In *AAAI*, volume 4, page 12.

Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. 2014. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328.

Matthew D Zeiler and Rob Fergus. 2014. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer.

Sven Meyer Zu Eissen and Benno Stein. 2004. Genre classification of web pages. In *Annual Conference on Artificial Intelligence*, pages 256–269. Springer.