September 2022

A WORKFLOW FOR DOCUMENT TEMPLATE EXTRACTION USING MACHINE LEARNING

HP INC

A workflow for Document Template Extraction using Machine Learning

Abstract

This disclosure aims to allow the user the possibility to extract the template of a document using only a picture of it. Our method can be described as follows: 1. The document image is identified in the scene (when mobile scanning, for instance); 2. The document image is cropped from the scene; 3. The document image is segmented into its different regions: e.g., title, image, graphic, content text; 4. For each region, an algorithm (e.g., a ML model) will be used to extract the region template features and an OCR engine will extract the text from the region; 5. The template features will be processed by another algorithm capable of a. matching those “unknown” features to known ones, or b. return a template code format (e.g., Latex format); 6. The template regions and extracted texts are joined in a customizable software (e.g., Microsoft Word or TeXstudio), so the user can modify it.

Problems Solved

Extracting the template of a document image allows the user the capability of both reusing it with his own information or adjusting it to his own purpose. For instance, an old business card can be scanned and its template can be extracted, so the user can update it without the need of recreating the original template. Moreover, it can be used to facilitate the document template creation process, since it is possible to base it from another one.

Proposed Solution

Our solution goes as follows:

1. The document image is identified in the scene (when mobile scanning, for instance);
2. The document image is cropped from the scene;
3. The document image is segmented into its different regions: e.g., title, image, graphic, content text;

4. For each region, an algorithm (e.g., a ML model) will be used to extract the region template features and an OCR engine will extract the text from the region;

5. The template features will be processed by another algorithm capable of a. matching those “unknown” features to known ones (as described in the *appearance matching* approach), or b. return a template code format, e.g., Latex format (as described in the *code style matching* approach);

6. The template regions and extracted texts are joined in a customizable software (e.g., Microsoft Word or TeXstudio), so the user can modify it.

The first and second steps are easily achieved using, for example, with the Fan algorithm [1] and can be bypassed when the document is scanned using some dedicated hardware, such as a flatbed scanner.

The third step can be also returned by an algorithm such as [1], since it segments the document into regions and determines the type of each one, as explained by Virgil Russon [5]. Also, other techniques such as the one proposed by Hui Chao and Jian Fan [2] could also be employed here.

With the segmented regions, any OCR engine can be used to extract the text (e.g., Tesseract). The main challenge now is to retrieve the region template. Below are two solutions for this problem that, although relies on different approaches, can be implemented in similar ways.

Appearance matching. In this approach the segmented regions will have their templates returned from an appearance matching with pre-trained templates. This can be achieved using a feature extractor algorithm (e.g., an Autoencoder ML model) and a matching technique (e.g., KNN) to match the extracted features from the regions of a scanned document to an existing one stored in a database. There are many methods related to how to extract features, but the core idea keeps the same: extract the most important features that are capable of better describing the input data. For instance, an AutoEncoder is a non-supervised ML model that is composed of an encoder and a decoder module capable of extracting such features. The encoder reduces the dimensional size of its input while stacking image features in its channels. The decoder then uses the stacked features from the encoder to reconstruct the input image. That way, the encoder learns which features to extract from its input.

Hence, an AutoEncoder model can be trained receiving many different known template formats (i.e., template formats that we can decode the image to a modified version of it, e.g., using Microsoft Office or TeXstudio code language) so it can learn the

main features from these different template images. These learned features will be stored in a database containing pairs of extracted features and their respective template. Now, in order to match a new template region (unknown) to a known one, the following steps can be taken:

1. Feed the encoder module with the image template and collect the extracted features of it;
2. Match the extracted input template feature with the closest feature in the database, using, for example, the KNN algorithm;
3. Use the known template information to provide a modifiable version of it to the user.

Notice that each of these steps can be done both in the whole image and hence returning a general template that better fits to the input image; or it can be done in a region by region fashion, i.e., each detected region on the document image (e.g., text box, image, graphic) will be matched to a region template and, at the end, all these regions will be joined based on their spatial displacement on the original input image. In other approaches, it can also use both the information for the whole document image and the individual regions. This way, we can have a global template (for the whole image) and local templates (for each region) and a merge algorithm can be used in order to "average" the contribution of the local templates to the global one.

Code style matching. This second approach can be harder to implement, but it is capable of returning more reliable results. In the Appearance matching approach, we are trying to match the segmented region to a previous (already seeing by the model) template. Hence, there can be limitations regarding the usability of the algorithm related to the size of the templates database. With the code style matching, we overcome this problem by directly inferring the template format based on some textual coded language, such as LaTex [3].

The more popular text editors are conceived as "you get what you see", i.e.: what you write in the text editor is what you will be printing. However, there are editors where you can directly act on the text output by coding its appearance. That is what happens when using a LaTex based editor, for example. Just to clarify the difference, a Word bold text can be seeing by the user as:

A bold text.

While in LaTex:

```
\textbf{A bold text.}
```

Hence, if we can code the output, we can infer the whole template of the region.

The process of outputting a text from an input image is called *Image Captioning* [4] and it is achieved by mixing Natural Language Processing (NLP) and Computer Vision (CV) techniques. The idea of these techniques is very similar to the AutoEncoders one: the features extracted by the input images are matched with a NLP model that outputs the text. That way, the matched features will not be regarding a pre-defined seen image, but a code-style that can be way more general.

Finally, after identifying the region templates, they can be all merged with their corresponding texts to form the final document. This final document will then be completely editable by the user, so he can change the found templates or texts. It is important to notice that each found template for each document region will also be placed on the position where the document region was found.

References

[1] Fan, J. (2007). Enhancement of camera-captured document images with watershed segmentation. CBDAR07, 87-93.

[2] Chao, Hui, and Jian Fan. "Layout and content extraction for pdf documents." International Workshop on Document Analysis Systems. Springer, Berlin, Heidelberg, 2004.

[3] LaTex. Available at: https://www.latex-project.org/

[4] Image Captioning in Deep Learning. Available at: https://towardsdatascience.com/image-captioning-in-deep-learning-9cd23fb4d8d2

[5] Type Determination using a Block Weight. HPICS 2018. http://hpics.corp.hp.com/1sw/hPICS18/abstracts/111.pdf

Disclosed by:

Lucas Nedel Kirsten and Erasmo Isotton

HP Inc.