A complete approach to automating document information extraction

Business objective
Process Optimization
Sector
Bank and Insurance

More and more, organizations apply Machine Learning and related algorithms to the automated processing of documents, a field otherwise known as Optical Character Recognition (OCR).

If you have to deal with high quantities of visual or textual files, you can spare a lot of time and expenses using ML and automation found within the right tools instead of manually handling the files.

This article covers the approach taken in one of our projects for the **challenge of document classification and information extraction**. In the context of this project, our client wanted to insert an automated process to classify documents in their already well-developed operations ecosystem. The objective was to receive PDF scans filled manually by customers, retrieve all necessary information, and send it to the client.

The solution for document information extraction

We created a pipeline composed of a series of checks and processing steps. Each PDF went through this pipeline to be fully processed.

First, we recognized the current template among a set of possible templates. The template is recognized by its front page, containing a header and usually other special features. These features make the differentiation easier in *OpenCV*. We use the function `cv2.matchTemplate` to get a correlation score between the current document's first page and all the first pages of the database of templates. If the best candidate has a correlation score higher than a minimum threshold, the template is considered as *recognized*.

We housed the database of templates and their configuration files on a bucket S3 on *AWS,* queried from inside a *Docker* image containing the pipeline.

After this, each page of the current document we needed to process had to link to its matching page of the identified reference template. The use of that step is to ensure that the algorithms know on which page they are working. This is crucial to determine which information corresponds to which question on the document. The pipeline compares the first page of one reference document to the first page of another in treatment, and so on. If there are missing pages or unordered scans, the pipeline sends the document for manual review. Comparison page by page uses the same principle as template recognition, with heavy use of our dear `cv2.matchTemplate`.

Once we identify the pages, the information boxes of each page are then automatically detected. The detection is done by mainly using the functions `cv2.morphologyEx` and `cv2.connectedComponentsWithStats` from *OpenCV*. `cv2.morphologyEx` to apply a horizontal and vertical edge filter on the pages. And, `cv2.connectedComponentsWithStats` finds the closed boxes created by the vertical and horizontal edges and returns their positions.

After, one more filtering is applied to keep the boxes fulfilling particular position and dimension constraints. From the final positions, the pipeline crops the information boxes from the main page image. The boxes, in our case, are containing the answers to the questions of the document. The pipeline specifies constraints in the config file of the template.

If the pipeline functions did not detect all information boxes on a page, then the step is assumed to be failed, and the pipeline sends the document for manual review.

This pipeline sends cropped images to a Convolutional Neural Network (CNN). The network of neurons already trained decides each checkbox and associates it with a confidence level. The confidence level is the probability associated with the label of the content of the box. If it is under a certain threshold probability, it means the model is unsure about the content, and the pipeline sends the document for manual review.

If all the previous steps have succeeded, the pipeline sends the answers to the client in a JSON format.

The challenges during document information extraction

We met a few challenges during the development; the variety of templates, the variety of human behavior, and the hard reliability requirements.

Variety of templates

First, for the company, there are about ten different templates currently in use. To have good coverage, the pipeline had to be able to recognize a few of them. We opted for the two most common ones, who make 90% of all documents. Moreover, for each template, we needed a configuration file for the algorithm to link the detected boxes to their question specifications.

The solution: Because this configuration can be complex for a non-initiated user, we created a *template generator* to add any new template in a few clicks.

Variety of human behavior

Another challenge was to realize the variety of human behavior in filling documents and try to cope with it. We had to find the trade-off between maximizing the coverage of automation and minimizing the process's complexity. A complexity that increases with the more particular cases you want to cover.

Moreover, especially for the classification of boxes with the Neural Network, it is impossible to ensure 0% error in the current framework. Because the documents were not initially created to be automatically treated, customers sometimes fill their answers in the most interesting way.

The solution: There is no correct answer to find a good trade-off. Depending on the client, you could spend more time developing a highly complex infrastructure to process almost 100% of documents, therefore maximizing the coverage. But the cost of creating such an infrastructure would not be worth the gain. Usually, a client will want to optimize his *Return On Investment* (*ROI*), keeping in mind that paper documents are getting rarer with the all-digital era.

Concerning the filling of the boxes, the only way to cope with it is to install a clean framework where we explain the method of filling to the customers. That framework should appear on future templates.

A hard requirement in reliability for document information extraction

Finally, we had to meet a hard requirement in reliability, ideally 0% of error. In machine learning terms, we had to assure 100% precision, which is the consequence of depleting the recall, a.k.a. the coverage.

High reliability is necessary for our clients to have complete trust in the automation and integrate it into their operations. However, this has the consequence of diminishing the number of documents that we can automate.

The solution: For a given architecture, the typical way to find the best pipeline with this 100% precision while maximizing the coverage is to do hyperparameters optimization. Using the dataset of previous documents, one can tweak the parameters at each step while fulfilling the two conditions. The parameters here are mainly the thresholds of correlation for template and page recognition and the confidence threshold of the neural network. If we choose a very harsh threshold, we will assure 100% precision, but the coverage will deplete. If we choose it too loosely, errors will arise, and 100% precision is lost. We developed two versions.

The tools for document information extraction

We used a series of tools to implement such a pipeline and put it into full production, namely OpenCV, PyTorch, Docker, and AWS.

OpenCV

We made heavy use of the *OpenCV* library at different steps of the process. *OpenCV* is a known open-source library for classical computer vision. We used it for template recognition, page order checking, and automatic detection of the information boxes.

PyTorch

We developed the neural network model in *PyTorch*. It is a Convolutional Neural Network (CNN), perfect for image classification. Transfer learning was applied by starting from the pre-trained ResNet50 architecture before training on the client's images. The CNN is giving the class and a probability score on how confident it is about its choice. This score is helpful to ensure no misclassifications.

Docker

We deployed our pipeline in a container using *Docker*. The real advantage behind it is to stop worrying about compatibility issues when deploying to new servers. The process runs in the "black box" container and works on any machine having *Docker* installed.

Amazon Web Services (AWS) and *the Cloud Development Kit*

Our client already had multiple services in a stack deployed on *AWS*. We ensured to fit this project with the rest of their ecosystem. In practice, it means that the operations are running daily on the servers of *Amazon*, rented by our client.

The advantage of using the *Cloud Development Kit* is that all the infrastructure can be specified as code and deployed in the command line. The main components used are *Buckets*, *Lambda* functions, and the *Elastic Container Service* and *Batch*. All this allows to deploy *Docker* containers and adapt flexibly to the number of documents to process by allocating more or fewer servers.

AWS also offers a tool to do human image labeling on big datasets efficiently. It was instrumental to train the Neural Network on the client's data. The tool dispatches the labeling to different persons who will assign the proper label to the image.

Conclusion

In the final version of our solution, the client could automate **47% of all documents** coming to the pipeline after the document information extraction. It's safe to say this saved the client a considerable amount of time and expenses.

Gain estimation for document information extraction

Our client receives 100 documents each day on average. Each document takes 75 seconds to be manually processed. The client estimates that it costs about **26.000 €** per year. Given the pricing of *Amazon* servers, our installation cost should be less than 100 € per year, which is negligible. Because we automate ~50% of the documents, the spared amount per year is **~13.000 €**.

The other impact is that because we made the architecture with modularity, it could be with lesser cost extended to different types of client documents, where there is information extraction involved.

Limitations and possible improvements

The pipeline will work if the document is from a suitable template, with pages in the correct order, and the document has correctly completed information in the dedicated boxes.

Another possible risk that can arise in such machine learning processes is the evolution of data. This is called dataset shift. If the types of templates change, if the quality of the pdf scans deteriorates, the coverage could drop. To address such a problem, reporting daily performances must be put in place to measure any sizable difference in results.

Ideas to improve the automation coverage would be to create better-filling guidelines in the document and only keep one good template for the document.

For the current architecture, two direct improvements could give better results: improve the boxes' automatic detection (only 75% that could come close to 100%) and an automatic reordering of the pages of an unordered document.

Finally, to ensure a significant *ROI*, the ideal would be to extend the architecture to all documents still treated manually containing information boxes.

About Agilytic

Since 2015, Agilytic helps innovative leaders solve their biggest challenges through the smarter use of data. With over 150 successful projects to date, we have perfected a pragmatic approach to putting data at the service of business goals, be they commercial, operational, financial, or human. Reach out today for a quick introduction, we'd love to hear from you.