CAPSTONE PROJECT Adamou Nchange Kouotou

Machine Learning Engineer Nanodegree April 11th, 2021

An optical data entry tool for KYC and customer authentication

Definition

Project Overview

Verification of identity is a tedious and recurrent task that employees of financial services companies have to perform on daily basis. Before providing a service to a customer, any financial service agent has to perform KYC procedure if it is a new customer or proceed to authentication if it is an already registered customer. For that sake, they have to manually enter on a software the information that are on the identity document provided by the client. The tedious and boring nature of this task always results in poor performance of those agents, thus leading to poor quality of data into customer databases of these companies.

In this project, I created an App that leverages Tesseract, an OCR library, to automatize extraction of information on identity documents. OCR (Optical Character Recognition) is a computer vision technique used in data science to convert image texts into machine-encoded texts.

Problem Statement

The goal is to create an identity document information extractor running on Android smartphones. The app must be able to detect if a document has a MRZ (Machine-readable-zone) and if it is the case, extract identity information from it. The tasks involved are the following:

- Create a dataset of images of different types of identity document.
- Write an OCR model (code) that can extract identity information from an MRZ image with high accuracy.
- Make an Android App that leverage the model above to extract identity information from an identity document.

We expect our code to have a better accuracy than Passporteye text extraction function called ***read_mrz()***.

Metrics

Per-Character Recognition rate (PCR) is a metric which is commonly used to evaluate performance of OCR systems.

$$PCR = \frac{Number\ of\ correct\ characters}{Adjusted\ dataset\ size}$$

I used this metric when evaluating the model because the greater the number of correct characters the better the experience of the user; and the metric has to take into account any additional unexpected characters that might deteriorate the quality of the extracted text.

- A correct character is a character that is correctly extracted from the MRZ image and that occupy its right position.
- The Adjusted dataset size is the length (number of characters) of the longest text between the original MRZ text and the machine extracted text. The Adjustments of the size of the dataset size is justified by the fact that we must consider an additional empty character (“”) for each unexpected additional character generated by the machine to take into account their impact on the PCR measurement.

Analysis

Data Exploration

As I leveraged Tesseract library for Optical Character Recognition, I did not need to train a model on a dataset. I used a dataset only to measure performance of my improved design of MRZ reader. The dataset that I used can be found in this Github repo.

This dataset is composed of two video of a Cameroonian passport that a recorded myself with an android smartphone and some video of identity documents that I downloaded from the MIDV-500 dataset.

Exploratory Visualization

The characteristics of the dataset used in our experiments are the following:

Identity document Type	Number of videos
Cameroonian passport	2
Brazilian passport	10
Total	12

Fig. 1: Dataset structure

Each identity document that we have a video in our dataset contains a MZR (Machine-readable-zone). MRZ is the main region of image of identity documents that our model is going to examine. The MRZ of an identity document is something that most people take little to no notice. However, if you look closely at it you will see that it contains the same information about the document owner as is shown on the rest of the identity page.

Fig. 2: Photo of a brazilian passport in a natural environment

Fig. 3: Photo of a cameroonian passport in a natural environment

Algorithms and Techniques

I used Passporteye ***read_mrz()*** function to extract identity information from identity documents. However, our initial trials with that approach led to poor results. It could hardly correctly read more than half of the characters of the MRZ. Moreover, when we use Passporteye on different photos of the same identity document, it outputs different results.

To solve this problem, I implemented on top of Passporteye ***read_mrz()*** function a code that improved the overall performance of our OCR model. I developed three functions that constitutes the main components of our methodology to improve the PCR of our proposed OCR model.

i) Maximum likelihood of characters

This is a statistical approach which consists of generating different sizes of the same images and then use ***read_mrz()*** function to extract identity information from each of those images.

Then, consider for a given character position the character that appears the most on that position from the set of extracted results.

ii) Text-field parsing

I developed a parsing code which aims to filter the useful information from the extracted text that it takes as input.

Some main features of this parsing code are listed below:

- The code replaces the character “<” by “” as that character is used for fields separation on MRZs and therefore it be considered as part of an identity information.
- The code splits its inputs by the double-space character (“ “) and it only keeps the first cell of the resulting list as a in the case of a name or a surname composed of two words, the two words are always separated by a single-space instead of a double-space.
- The code splits its inputs by single-character words and only keep the first cell of the resulting list as names are composed of at least two letters.

iii) Check-digits verification

The MRZs of identity documents contains some digits (characters) called check-digit. These digits are results of mathematical computation of the other characters of the same MRZ. I developed a function that takes advantage of this intrinsic feature of MRZs of identity documents.

For each improved text extracted from an MRZ, this code computes the check-digits for **document number, date of birth** and **expiration date** fields MRZ. Then it compares the results to the values of the check-digits extracted on the same document. We repeat this operation with different set of transformation of the same image of identity document until we obtain a perfect match.

Benchmark

According to its documentation, Passporteye precision on identity information extraction from MRZs is around 80%. However, while testing Passporteye on our dataset, I obtained a less good result. The table below shows PCR results of Passporteye on our dataset.

	Passporteye read_mrz() function		
	Cameroonian passport	**Brazilian passport**	**Overall dataset**
PCR	0.66	0.56	0.61

Fig. 4: Passporteye Per-Character Recognition rate on our dataset

Methodology

Data preprocessing

To build this dataset, I recorded a video of a Cameroonian passport with my smartphone and then I converted it into a set of images thanks to ***VideoCapture()*** and ***imwrite()*** methods of OpenCV library. I used the same function to convert into images some video of other identity document that I downloaded from the MIDV-500 dataset.

Then, I manually delete images that had incomplete or blurry MRZ from our dataset. This final preprocessing do not distort our experiment goal as our experiment aims to measure the capacity of models to read a text but not to guess it.

The table below presents the structure of our preprocessed datatset.

Identity document Type	**Number of images**	**Dataset size (Total number identity information characters)**
Cameroonian passport	69	55
Brazilian passport	79	49
Total	148	N/A

Fig. 5: Dataset structure

Implementation

The implementation process can be split into two main stages:

- The model development stage
- The application development stage

a) Model development stage

During the first stage, I wrote a code that leverage Passporteye ***read_mrz()*** function to extract identity information from an image of an identity document. To improve the precision of that code, I developed three functions implementing the following precision improvement processes:

- Maximum likelihood of character;
- Text field parsing;
- Check-digits verification.

The algorithm that I implemented are explained in the diagrams below:

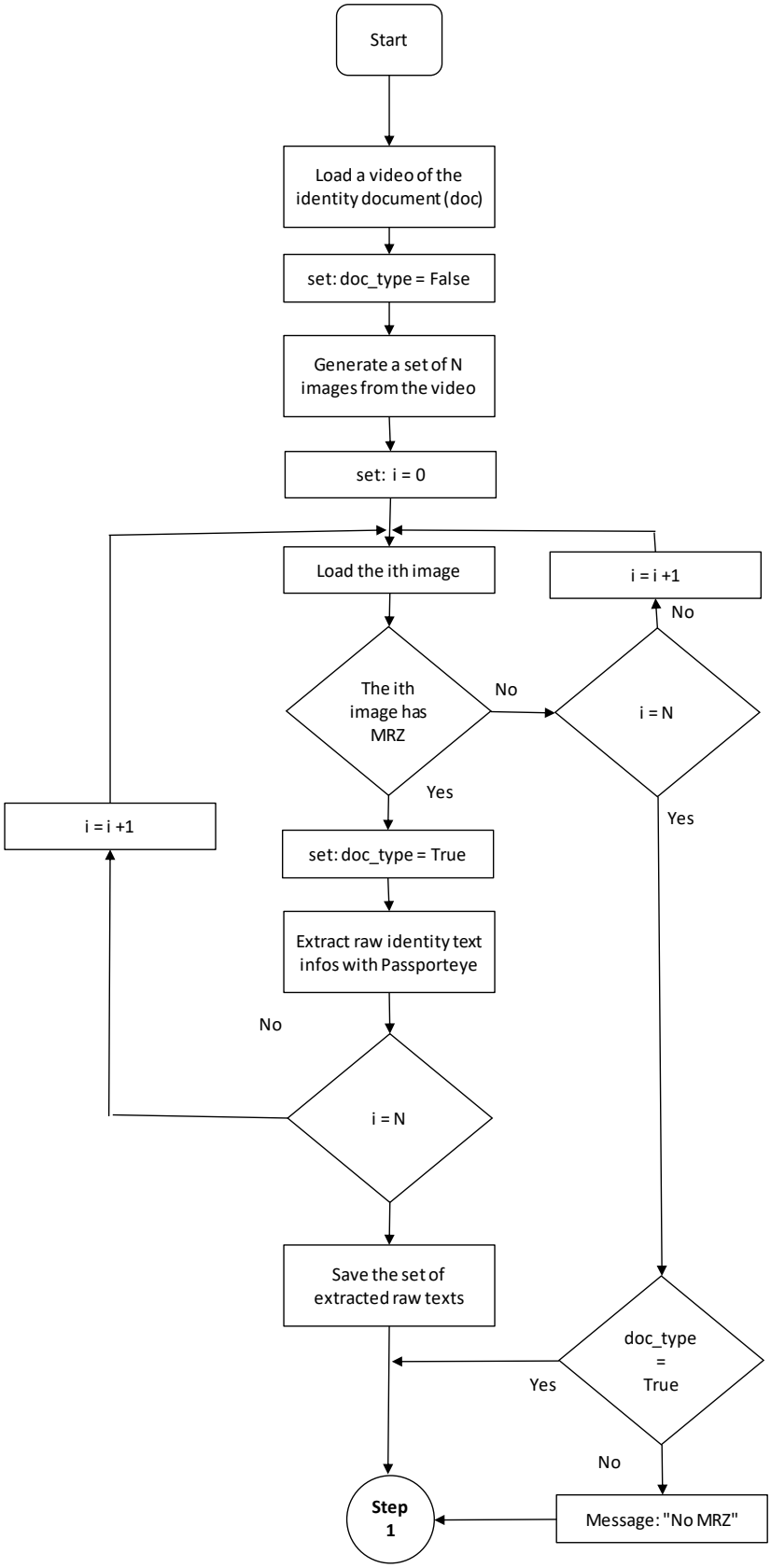

Fig. 6: Algorithm to extract a set of raw identity text infos

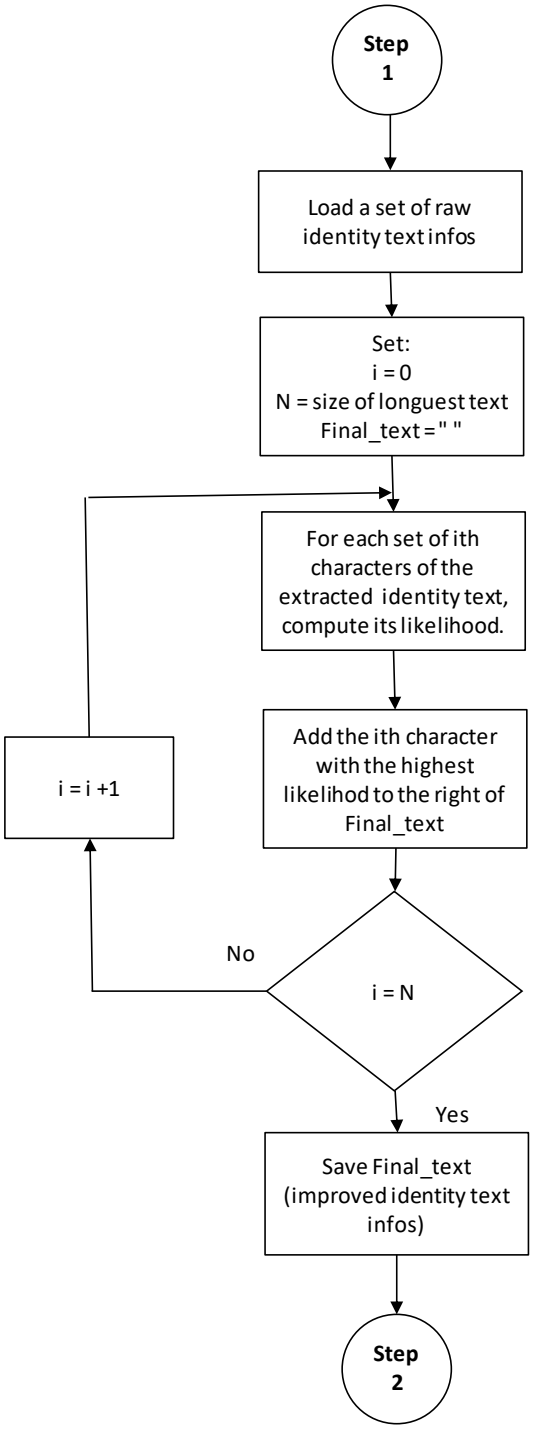

Fig. 7: Algorithm of Maximum likelihood of character

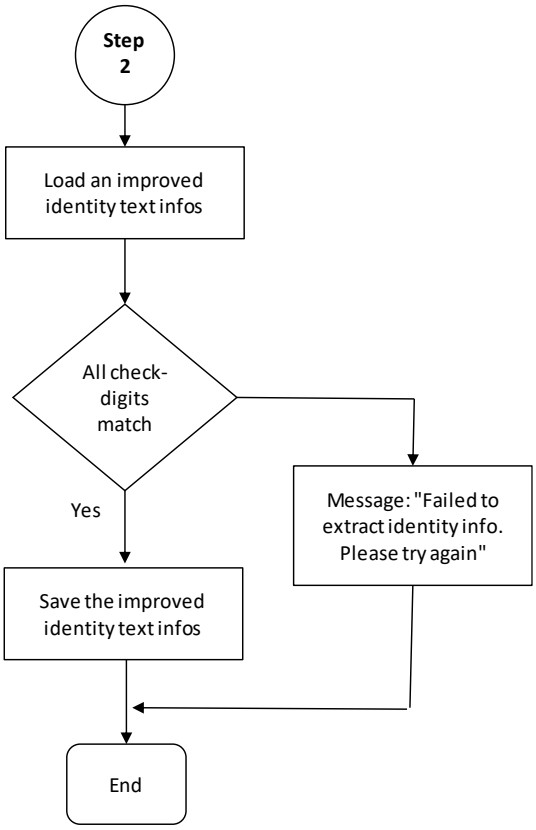

Fig. 8: Algorithm of Check-digits verification

b) Application development stage

The application development stage can be split into the following steps:

1. Development of the client app (mobile app / android app)
2. Development of a Rest API (FastAPI) to serve as interface with our OCR model
3. Interconnect the app to the OCR model via the API

Software requirements

To implement the proposed solution, there are some software requirements. The requirements listed below are those that I found on documentations of libraries and tools that I used.

- Pyhton 3 or higher (I used Python 3.8.5)
- OpenCV 3 or Higher (I used OpenCV 3)
- Pytesseract (I used Pytesseract 4.1.1)
- Passporteye (I used Passporteye 2.1.0)
- Uvicorn (I used Uvicorn 0.13.4)

a) Installing Python 3 and common libraries

To install Python 3 and all its common libraries, I recommend to install Anaconda which is a distribution of Python and R programming languages that simplifies packages management and deployment. Detailed instructions to install Anaconda can be found on this link.

b) Installing OpenCV 3

OpenCV is a library of programming functions mainly aimed at real-time computer vision. Detailed instructions to install OpenCV can be found on this link.

c) Installing Pytesseract

Pytesseract (Python-Tesseract) is an optical character recognition (OCR) tool for python. Detailed instructions to install Pytesseract can be found on this link.

d) Installing Passporteye

Passporteye is a package that provides tools for recognizing machine readable zones (MRZ) from scanned identity documents. Detailed instructions to install Passporteye can be found on this link.

e) Installing Uvicorn

Uvicorn is a lightning-fast ASGI server implementation, using uvloop and httptools. Detailed instructions to install Pytesseract can be found on this link.

Refinement

As mentioned in the benchmark section, Passporteye **read_mrz()** function achieved an precision of only 61% on our dataset.

This was improved upon to obtain our OCR model by implementing on the top of Passporteye **read_mrz()** function the following techniques:

- Maximum likelihood of character;
- Text-fields parsing;

While “Check-digits verification” technique does not improve the quality of the text that it receives and thus cannot be fully consider as an improvement technique, it can be used as a precision threshold for “Maximum likelihood of character” and “Text-fields parsing” algorithms.

Thus, to develop our identity information extraction app, we can use “Maximum likelihood of character” and “Text-fields parsing” techniques to refine the information extracted from the identity document and “Check-digits verification” technique to set the threshold at which we consider that the result produced by the app is acceptable.

Results

Model Evaluation and Validation

The diagram below depicts the PCR that we obtain at each step of our improvement process of our model.

	Per-Character Recongnition (PCR)			
	Passporteye		**Passporteye + Maximum likelihood of character**	
	Without parsing (Benchmark)	With parsing (Step 1)	Without parsing (Step 2)	With parsing (proposed OCR Model)
Cameroonian passport	0.66	0.75	0.77	**1.00**
Brazilian passport	0.56	0.80	0.60	**0.88**

Fig. 9: Passporteye PCR vs proposed OCR model PCR

The result displayed in the above diagram results from an experiment that I conducted with a set of 5 images. It shows that the implementation of our “Maximum likelihood of characters” algorithm on top of Passporteye drastically improves quality of the information extracted from identity documents.

Note that I conducted the same experiment with size of images set which were greater than five and I obtained approximately the same value of PCR for the “Maximum likelihood of character” algorithm. However, with size of images set that were smaller than five, I obtained a lower value of PCR for the “Maximum likelihood of character” algorithm.

Finally, the better PCRs of these models on the Cameroonian passport can be explained by the fact that in our experimental dataset, Cameroonian passport images where of better quality (clearer MRZ) than Brazilian passport images.

Justification

The maximum likelihood of character is the process that mostly improved the precision of our model. This is not a surprise because according to the law of big numbers, the characteristics of a random sample approximate the statistical characteristics of the population (set of individuals or elements) when the sample size increases. In our case, our

sets of population were the characters of a given position in the extracted texts and the statistical characteristics were the actual character in the identity document.

The field-text parsing algorithm was inspired by the structure of the extracted text that we obtained at the intermediary steps of our experiment. Exploring those intermediary results allow us to identify some weaknesses of Passporteye and correct those using a parsing approach.

Finally, it is an evidence that the fact of embedding inside our model a verification rule for the extracted information using check-digits would improve the app precision. Even if some information such as name and surname doesn't have check-digits on an MRZ, it is an evidence that as the precision of information that have check-digit increase, those of name and surname also increase because our improvement techniques are applied to the whole MRZ information.

Deliverables

To make this work practical, I developed an android app that leverages the proposed OCR model and the check-digits verification to extract identity information from identity document. The app is not already perfect, but it is a good start for the development of a professional tool.

Experiment source codes

The source code (experiment.py) of the experiment can be found on Github on this link.

1. Clone or download the Github repo of the source code into the server computer
2. Start your Anaconda or Python Terminal
3. Use cd command to navigate to the folder containing the file experiment.py
4. Set the path to the image folder and the validation data on the code of the **experiment.py** file at line 164 and 166 (see figure. 10)
5. Execute on the Terminal the command: **python experiment.py**

```
161                 }
162     """
163     # Set the path to the folder containing images to use for the experiment
164     images_folder = "data/brazil"
165     # Set validation data
166     valid_mrz = actual_mrz_bra
167     # Run the experiment
168     for i in [3,4,5,6,7]:
169         print(f"\n***RESULT FOR A SAMPLE SIZE EQUAL TO {i} ***")
170         unit_experiment(images_folder, valid_mrz, i)
171         print('*****************************')
172
```

Fig. 10: Setting experiment parameters

API source code

The source code (api.py) of the app can be found on Github on this link.

To test the API, follow the step below:

1. Clone or download the Github repo of the source code into the server computer
2. Start your Anaconda or Python Terminal
3. Use cd command to navigate to the folder containing the file **api.py**
4. Execute on the Terminal the command: **uvicorn api:app --host xxx.xxx.xxx.xxx** , where xxx.xxx.xxx.xxx represents the IP address of the computer hosting the API.
5. Open on you navigator the url: http://xxx.xxx.xxx.xxx:8000/docs
6. The Swagger interface below will appears and let you test the API.

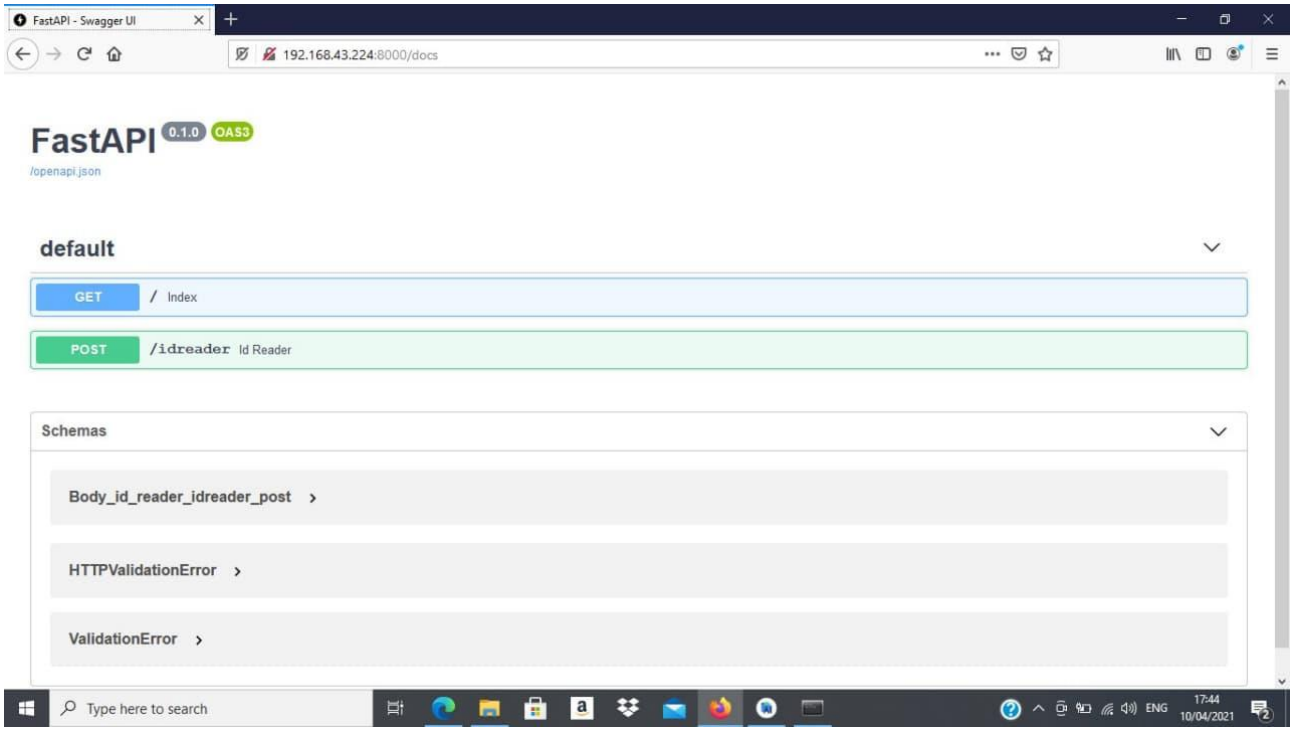

Fig. 11: API test interface on Swagger

App source code and executable file

The work is still in progress to finalize the Android app development. You can follow the development of the Android app on the Github repo (the source code of the app is in the folder named **idinfoextractor**).

To test the Android app at its current status.

1. Clone or download the Github repo of the source code into the server computer
2. Open the project on Android Studio
3. On the file named **UploadUtility.kt** at line 20, change the **serverURL** variable by your API URL (http://xxx.xxx.xxx.xxx:8000/idreader, where xxx.xxx.xxx.xxx represents the IP address of the computer hosting the API) (see figure. 12)
4. On the file named **UploadUtility.kt** at line 21, change the **serverUploadDirectoryPath** variable by your server URL (http://xxx.xxx.xxx.xxx:8000, where xxx.xxx.xxx.xxx represents the IP address of the computer hosting the API) (see figure. 12)

Fig. 12: Setting App connection to api.py file hosted on a web server

5. Build the app
6. Enjoy! ☺