PassportEye Documentation

Release 1.0.1

Konstantin Tretyakov

May 31, 2017

Contents

The PassportEye package provides tools for recognizing machine readable zones (MRZ) from scanned identification documents. The documents may be located rather arbitrarily on the page - the code tries to find anything resembling a MRZ and parse it from there.

CHAPTER 1

Installation

The simplest way to install the package is via `easy_install` or `pip`:

```
$ pip install PassportEye
```

Note that *PassportEye* depends on *numpy*, *scipy*, *matplotlib* and *scikit-image*, among other things. The installation of those requirements, although automatic, may take time or fail sometimes for various reasons (e.g. lack of necessary libraries). If this happens, consider installing the dependencies explicitly from the binary packages, such as those provided by the OS distribution or the "wheel" packages. Another convenient option is to use a Python distribution with pre-packaged *numpy*/*scipy*/*matplotlib* binaries (Anaconda Python being a great choice at the moment).

In addition, you must have the Tesseract OCR installed and added to the system path: the `tesseract` tool must be accessible at the command line.

CHAPTER 2

Service usage

On launching server/index.py (or using our provided Docker image), PassportEye is going to run as a service on 5000 port, exposing an API that can be used from your services to verify documents. *The API is documented here*.

CHAPTER 3

Command-line usage

On installation, the package installs a standalone tool `mrz` into your Python scripts path, which can be used from command-line for batch verification of documents and for other purposes, as necessary. *The command-line API is described here*.

CHAPTER 4

Python usage

You can also use MRZ as a library in your Python code if you want to be able to integrate it deeper and have better control over features present. *Python usage is described here*.

How PasswordEye can be used

PasswordEye can be used from command-line, through its service API or through its Python API:

Python usage

In order to use the PasswordEye functions in Python code, simply do:

>> from passporteye import read_mrz >> mrz = read_mrz(image_filename)

The returned object (unless it is None, which means no ROI was detected) contains the fields extracted from the MRZ along with some metainformation. For the description of the available fields, see the docstring for the *passporteye.mrz.text.MRZ* class. Note that you can convert the object to a dictionary using the `to_dict()` method.

If you want to have the ROI reported alongside the MRZ, call the `read_mrz` function as follows:

```
>> mrz = read_mrz(image_filename, save_roi=True)
```

The ROI can then be accessed as `mrz.aux['roi']` – it is a numpy ndarray, representing the (grayscale) image region where the OCR was applied.

For more flexibility, you may instead use a `MRZPipeline` object, which will provide you access to all intermediate computations as follows:

```
>> from passporteye.mrz.image import MRZPipeline
>> p = MRZPipeline(filename)
>> mrz = p.result
```

The "pipeline" object stores the intermediate computations in its `data` dictionary. Although you need to understand the underlying algorithm to make sense of it, sometimes it may provide for insightful visualizations. This code, for example, will plot the binarized version of the original image which is used in the algorithm to extract ROIs alongside the boxes corresponding to the extracted ROIs:

```
>> imshow(p['img_binary'])
>> for b in p['boxes']:
..     plot(b.points[:,1], b.points[:,0], c='b')
..     b.plot()
```

After getting a response, *refer to our response description*.

Command-line usage

In command-line, running:

$ mrz <filename>

will process a given filename, extracting the MRZ information it finds and printing it out in tabular form. Running `mrz --json <filename>` will output the same information in JSON. Running `mrz --save-roi <roi.png>` will, in addition, extract the detected MRZ ("region of interest") into a separate png file for further exploration. Note that the tool provides a limited support for PDF files – it attempts to extract the first DCT-encoded image from the PDF and applies the recognition on it. This seems to work fine with most scanner-produced one-page PDFs, but has not been tested extensively.

After getting a response, *refer to our response description*.

Service usage

Uploading files to PassportEye through API is simple. For example, here's Python code you can use to do that:

```
>>> import requests
>>> print("ok")
>>> files = {'file': open('passport_scan.jpg', 'rb')}

>>> r = requests.post("http://passporteye.domain/upload_file",
>>> r.text
```

After getting a response, *refer to our response description*.

PassportEye API response documentation

Here's an example response you can get from PasswordEye:

```
{
  "ela": {
    "max_diff": 6
  },
  "exif": {},
  "mrz": {
    "bounding_box": [
      [
        309.5091162700831,
        19.500000612510348
```

```
      ],
      [
        309.5091162700831,
        234.50171538138144
      ],
      [
        274.4909911642962,
        234.50171538138144
      ],
      [
        274.4909911642962,
        19.500000612510348
      ]
    ],
    "check_composite": "0",
    "check_date_of_birth": "2",
    "check_expiration_date": "4",
    "check_number": "6",
    "country": "LVA",
    "date_of_birth": "821212",
    "expiration_date": "170305",
    "method": "direct",
    "mrz_type": "TD1",
    "names": "ANDRIS",
    "nationality": "LVA",
    "number": "PA9992921",
    "optional1": "121282<88882<<<",
    "optional2": "<<<<<<<<<<<",
    "sex": "M",
    "surname": "PARAUDZINS",
    "type": "I<",
    "valid_composite": true,
    "valid_date_of_birth": true,
    "valid_expiration_date": true,
    "valid_number": true,
    "valid_score": 100
  }
}
```


The JSON schema is as follows:

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "title": "Image trust score and scoring data",
  "description": "This JSON contains trust score and trust scoring data that was␣
↪derived from an image passed to PassportEye",
  "type": "object",
  "properties": {
    "ela": {
        "type": "object",
        "description": "Error Level Analysis score data",
        "properties": {
            "max_diff": {
                "type": "integer",
                "description": "Error Level Analysis score. Currently, returns the␣
↪biggest peak from all ELA peaks encountered."
            }
        }
```

```
    },
    "exif": {
        "type": "object",
        "description": "EXIF data from the submitted image as key-value pairs. The␣
↪data is taken directly from the photo, so the contained keys can be different,␣
↪depending on what EXIF information is present.",
        "properties": {}
    },
    "mrz": {
        "type": "object",
        "description": "Decoded Machine Readable Zone data",
        "properties": {
            "bounding_box": {
                "description": "Coordinates for bounding box for MRZ",
                "type": "array",
                "items": [
                    [
                      {
                        "type": "array",
                        "description": "X and Y coordinates for top right corner of␣
↪bounding box",
                        "items": {
                          "type": "number"
                        }
                      },
                      {
                        "type": "array",
                        "description": "X and Y coordinates for bottom right corner␣
↪of bounding box",
                        "items": {
                          "type": "number"
                        }
                      },
                      {
                        "type": "array",
                        "description": "X and Y coordinates for bottom left corner of␣
↪bounding box",
                        "items": {
                          "type": "number"
                        }
                      },
                      {
                        "type": "array",
                        "description": "X and Y coordinates for top left corner of␣
↪bounding box",
                        "items": {
                          "type": "number"
                        }
                      }
                    ]
                ]
            },
            "check_composite": {
                "description": "Composite MRZ checksum as parsed from MRZ",
                "type": "string"
            },
            "check_date_of_birth": {
                "description": "Date of birth checksum as parsed from MRZ",
```

```
                "type": "string"
            },
            "check_expiration_date": {
                "description": "Expiration date checksum as parsed from MRZ",
                "type": "string"
            },
            "check_number": {
                "description": "Personal number checksum as parsed from MRZ",
                "type": "string"
            },
            "country": {
                "description": "Country code",
                "type": "string"
            },
            "date_of_birth": {
                "description": "Date of birth in DDMMYY format",
                "type": "string"
            },
            "expiration_date": {
                "description": "Expiration date in DDMMYY format",
                "type": "string"
            },
            "method": {
                "description": "",
                "type": "string"
            },
            "mrz_type": {
                "description": "MRZ type",
                "type": "string",
                "enum": ["TD1", "TD2", "TD3", "MRVA", "MRVB"]
            },
            "names": {
                "description": "First name",
                "type": "string"
            },
            "nationality": {
                "description": "",
                "type": "string"
            },
            "number": {
                "description": "",
                "type": "string"
            },
            "optional1": {
                "description": "Optional MRZ field that may or may not contain useful␣
↪information",
                "type": "string"
            },
            "optional2": {
                "description": "Optional MRZ field that may or may not contain useful␣
↪information",
                "type": "string"
            },
            "sex": {
                "description": "Sex",
                "type": "string"
            },
            "surname": {
```

```
                "description": "Last name",
                "type": "string"
            },
            "type": {
                "description": "MRZ document type field contents",
                "type": "string"
            },
            "valid_composite": {
                "description": "Composite checksum alidity",
                "type": "boolean"
            },
            "valid_date_of_birth": {
                "description": "Date of birth checksum validity",
                "type": "boolean"
            },
            "valid_expiration_date": {
                "description": "Expiration date checksum validity",
                "type": "boolean"
            },
            "valid_number": {
                "description": "Personal number validity score calculated ",
                "type": "boolean"
            },
            "valid_score": {
                "description": "The MRZ trust score calculated. Is calculated based␣
↪on valid_* fields, ",
                "type": "integer"
            }
        }
    }
  }
}
```

How PasswordEye works

PasswordEye uses a number of heuristics, algorithms and sanity checks to decrease the It's designed to thwart the technological methods of identity spoofing, as well as prevent inevitable human error while submitting the documents.

To improve the system performance, decrease the server load and increase responsiveness of services, the platform using PasswordEye can request it to perform or not to perform certain checks, depending on the necessary security level and desired system resource consumption.

Image manipulation detection

PassportEye employs the most efficient image manipulation detection techniques that cover 99.5% of all artificially-crafted and digitally-manipulated images. The checks listed all notify the platform about possible document falsification.

- **Verification of fonts** used on the documents, using a database of known fonts used in identification documents worldwide, as well as optionally using an external neural network trained on distinguishing between different fonts.
- Check for **photo presence** on the identification document. The check uses a neural network TODO: ask Christian to elaborate

- Check for **possible digital manipulation** in JPEG documents, employing Error Level Analysis. The ELA check detects whether the provided image has been edited by image manipulation programs, specifically, if regions of the image have changed considerably compared to other regions. The check is based on the fact that editing the image requires re-compressing the edited parts, thus changing the statistical distribution of compression artifacts. The Error Level Analysis check implementation uses convolutional neural networks to ensure robustness and reduce false negative rate, as opposed to more popular algorithms using geometrical and statistical features that tend to have high error margin in edge cases. Therefore, the PassportEye ELA check has a high weight in calculating the overall trust score.

Human factor verification

PassportEye is capable of detecting simplest ways to work around the document verification process, while remaining user-friendly and providing seamless integration with the platform using it. As the industry experience shows, it's easy to check if the provided document conforms to specifications, but it's harder to check if it conforms to reality.

PassportEye sanity-checks provided documents to notify about most common mistakes that are typically made by users, as well as check for workarounds which have been used to bypass most common document verification schemes. The checks included are:

- Verification that the **image provided has an actual identification document**, using a database of identification documents. As PassportEye is not supposed the, the validity score influence of this check is inversely proportional to the current size of PassportEye database.
- Verification against **social media profiles**, calculating profile validity score and accounting for it while calculating the overall trust score. Currently, PassportEye has Facebook, Google+, MySpace and Twitter integration.
- Verification of the **document expiration date**, comparing it to current date and asking the user to submit a valid document in case the provided document has expired.
- **Sanity-checks** for document **issue/expiration date**, notifying users about expired documents and not accepting them as valid. When the issue and expiration dates aren't realistic, it notifies the platform about possible document falsification.
- Checks for **file size, magic markers and resolution** of provided file to prevent possible DoS attacks on PassportEye service. The scenarios covered by this check include submitting artificially-crafted images that consume operating memory of the system, submitting files that are designed to be interpreted as malicious code, as well as submitting files of unnecessarily high resolution.
- Checks for **documents that already went through validation** and yielded unfavorable results, notifying the platform about such occasions. The scenarios covered by this check include customers that are trying to game the platform by submitting same document on different platforms, hoping that either the platforms in question are using inferior verification systems (or none at all), or that the verification system provides non-deterministic results.

MRZ verification

MRZ - machine-readable zone

PassportEye uses Machine-Readable Zones (further: MRZ) to check for the markers of document falsification.

Example MRZ on document

As MRZ encodes all the human-readable data present on identification documents, it's trivial for a computer to compare MRZ data to data that's written in human-readable form on the document, notifying about possible inconsistencies and calculating the error score

PassportEye implements MRZ manipulation interface which internally conforms to the ICAO Document 9303 (endorsed as ISO/IEC 7501-1). The library is capable of:

- Decoding MRZ
- Detecting MRZ type
- Decoding

How PasswordEye calculates trust score

PassportEye uses multiple factors in calculating the total score, including MRZ verification score, EXIF legitimacy score,

- MRZ score
- Compares birth date checksum in MRZ to the calculated checksum - 15/100 points
- Compares personal number checksum in MRZ to the calculated checksum - 15/100 points
- Compares expiration date checksum in MRZ to the calculated checksum - 15/100 points
- Sanity-checks name length - 10/100 points
- Sanity-checks surname length - 10/100 points
- Sanity-checks country code - 15/100 points
- Sanity-checks nationality - 10/100 points
- Sanity-checks miscellaneous fields - 10/100 points
- EXIF check - can only decrease trust score
- Starts with 100 points
- Remains on the same level if EXIF is blank
- Remains on the same level if EXIF contains info about a digital camera
- Checks if the EXIF fields contains info about image editing programs - -100/100 points