##############################################################################

To run server on localhost from terminal

##############################################################################

After installation of all the dependencies.

Open the terminal at directory - flaskRest (folder provided with this file)

And run the following commands

> export FLASK\_APP=app.py

> flask run

Url: http://localhost:5000/getDocumentDetails

Only one URL is required to be exposed, the input and output format is kept in JSON format, details are provided below.

##############################################################################

# Dependencies Required for Python

##############################################################################

Flask

OpenCV 2.4.x

pytesseract

editdistance

passporteye

numpy 1.11 or above

PIL

base64

nltk (word\_tokenize)

##############################################################################

# Dependencies Required for OCR

##############################################################################

Tesseract 3.x

##############################################################################

# Input format

##############################################################################

In order to process the documents 3 inputs (Document Type, Document Image, Reference Document data) are required, format of the inputs is kept in JSON format.

Following are the descriptions of these requirements:

1) Document Type :

JSON Key : “document\_type”

JSON Value Type : Integer

Description : The value indicates the type of document(Passport, EID, etc) uploaded and also the source of the upload(camera or file).

Following are the code expected as inputs to identify the document type and source:

|  |  |  |
| --- | --- | --- |
| Code | Document Type | Source |
| 101 | Passport | Camera Upload |
| 102 | Passport | File upload |
| 201 | EID | Camera Upload |
| 202 | EID | File upload |
| 301 | VISA | Camera Upload |
| 302 | VISA | File upload |
| 401 | Salary Letter | Camera Upload |
| 402 | Salary Letter | File upload |

2) Document Image :

JSON Key : “document\_image”

JSON Value Type : String

Description : This JSON pair holds the image of the document provided by the user in Base 64 format,

The image uploaded by the user must be converted/encoded to Base 64 format to pass across the server for further computation.

3) Reference Document Data :

JSON Key : “reference\_document\_data”

JSON Value Type : JSON List of Name Value Pairs.

Description : In order to verify the document, its information needs to cross checked with the details extracted from the passport document,

This JSON pair holds the information extracted from the passport (output of passport extraction) in the exact same format.

For calling Passport document extraction, this field is kept null.

Eg. of the input format:

{"document\_image" : "/9j/4AAQSk . . . . ZJRgAB", # Base 64 encoded Image.

"document\_type" : 201 , # Code indicating document type.

"reference\_document\_data" : { # Extraction results from passport document.

"country": "GBR",

"date\_of\_birth": "14 MAR 1967",

"expirydate": "13 DEC 2024",

"mrz\_type": "TD3",

"name": "JANET YVONNE PACHECO",

"passportno": "520656655”

} }

##############################################################################

# Output format

##############################################################################

Output consists of 3 entities, in JSON format : Document Details, Error, Match Verification.

Following are the descriptions for individual output entity:

1) Document Detail :

JSON Key : “document\_details”

JSON Value Type : JSON List containing name value pairs of all the extracted results

Description : The extracted result from the document are represented as name value pairs. These name value pairs may change depending on the document type.

2) Error :

JSON Key : “error”

JSON Value Type : JSON List containing two name value pairs : Code and Description.

Description : To provide a feedback if the document was not processed properly, an error message is provided. It contains two values:

Eg. "error" : { "code" : 1000, "message" : "Document not clear"}

If no error is encountered then this field is kept null.

Following is the description of errors along with the associated code:

Code| Description

0100: "Unable to process the document, please try again",

1000: "Looks like I can not recognize your Passport. please try again",

1001: "Sorry your Passport is not clear enough. please try again",

1004: "Your passport is already expired.",

1006: "Invalid Date of birth in Passport",

2000: "Looks like I can not recognize your EID. please try again",

2001: "Sorry your EID is not clear enough. please try again",

2004: "Your EID is already expired.",

2006: "Invalid Date of birth in EID",

3000: "Looks like I can not recognize your VISA, please try again",

3001: "Unable to retrieve information from VISA, please provide a clear image",

3002: "Visa does not contain applicant's passport number.",

3003: "Visa does not contain applicant's name.",

3004: "Could not find visa expiry date.",

3005: "Your visa is already expired.",

4000: "Could not retrieve salary information",

4001: "Salary letter cannot be more than 3 month old.",

4002: "Salary letter cannot be future dated.",

4003: "Salary letter does not contain applicant's name.",

4004: "Unable to retrieve salary from the Letter",

4005: "Unable to run name match in salary letter document analysis, error in parsing input data",

9001: "Parsing error, unable to parse JSON input",

9002: "Unexpected error occurred, unable process the document",

3) Match Verification :

JSON Key : “match\_verification”

JSON Value Type : JSON List containing name value pairs

Description : Contains a list of name value pairs where name represents entity which is being checked and the value is always Boolean indicating if there is match with details of the reference document

All the JSON names used are provided in the JSON\_NAMES.txt file along with this file.

Eg. of the output format:

{

"document\_details": {

"date\_of\_birth": "14 MAR 1967",

"eidno": "18496781627318",

"expirydate": null,

"mrz\_type": "TD1",

"name": "JANET YVONNE PACHECO"

},

"error": null,

"match\_verification": {

"eid\_name\_match": true

}

}

##############################################################################

# Note: Source Code Alteration

##############################################################################

Passporteye package source code is altered to customize to current requirements

After installing the passporteye package in the system, inside site-packages

Replace the passporteye/mrz/image.py code with the following:

##################################### BEGIN CODE ####################################

'''

PassportEye::MRZ: Machine-readable zone extraction and parsing.

Image processing for MRZ extraction.

Author: Konstantin Tretyakov

License: MIT

'''

from skimage import transform, io, morphology, filters, measure

from skimage import img\_as\_float

import numpy as np

import tempfile, os

from ..util.pdf import extract\_first\_jpeg\_in\_pdf

from ..util.pipeline import Pipeline

from ..util.geometry import RotatedBox

from ..util.ocr import ocr

from .text import MRZ

class Loader(object):

"""Loads `filename` to `img`."""

\_\_depends\_\_ = []

\_\_provides\_\_ = ['img']

def \_\_init\_\_(self, filename, as\_grey=True, pdf\_aware=True):

self.filename = filename

self.as\_grey = as\_grey

self.pdf\_aware = pdf\_aware

def \_imread(self, filename):

"""Proxy to skimage.io.imread with some fixes."""

# If input is opencv Image, converting it to Skimage.

if type(filename) is np.ndarray:

if len(filename.shape)!=2:

return None

img = img\_as\_float(filename)

return img

img = io.imread(filename, as\_grey=self.as\_grey)

if img is not None and len(img.shape) != 2:

# The PIL plugin somewhy fails to load some images

img = io.imread(filename, as\_grey=self.as\_grey, plugin='matplotlib')

return img

def \_\_call\_\_(self):

if self.pdf\_aware and type(self.filename) is not np.ndarray and self.filename.lower().endswith('.pdf'):

with open(self.filename, 'rb') as f:

img\_data = extract\_first\_jpeg\_in\_pdf(f)

if img\_data is None:

return None

else:

fd, fname = tempfile.mkstemp(prefix='pythoneye\_', suffix='.jpg')

try:

with open(fname, 'wb') as f:

f.write(img\_data)

return self.\_imread(fname)

except:

return None

finally:

os.close(fd)

os.remove(fname)

else:

return self.\_imread(self.filename)

class Scaler(object):

"""Scales `image` down to `img\_scaled` so that its width is at most 250."""

\_\_depends\_\_ = ['img']

\_\_provides\_\_ = ['img\_small', 'scale\_factor']

def \_\_init\_\_(self, max\_width=250):

self.max\_width = max\_width

def \_\_call\_\_(self, img):

scale\_factor = self.max\_width/float(img.shape[1])

if scale\_factor <= 1:

img\_small = transform.rescale(img, scale\_factor)

else:

scale\_factor = 1.0

img\_small = img

return img\_small, scale\_factor

class BooneTransform(object):

"""Processes `img\_small` according to Hans Boone's method

(http://www.pyimagesearch.com/2015/11/30/detecting-machine-readable-zones-in-passport-images/)

Outputs a `img\_binary` - a result of threshold\_otsu(closing(sobel(black\_tophat(img\_small)))"""

\_\_depends\_\_ = ['img\_small']

\_\_provides\_\_ = ['img\_binary']

def \_\_init\_\_(self, square\_size=5):

self.square\_size = 5

def \_\_call\_\_(self, img\_small):

m = morphology.square(self.square\_size)

img\_th = morphology.black\_tophat(img\_small, m)

img\_sob = abs(filters.sobel\_v(img\_th))

img\_closed = morphology.closing(img\_sob, m)

threshold = filters.threshold\_otsu(img\_closed)

return img\_closed > threshold

class MRZBoxLocator(object):

"""Extracts putative MRZs as RotatedBox instances from the contours of `img\_binary`"""

\_\_depends\_\_ = ['img\_binary']

\_\_provides\_\_ = ['boxes']

def \_\_init\_\_(self, max\_boxes=4, min\_points\_in\_contour=50, min\_area=500, min\_box\_aspect=5, angle\_tol=0.1, lineskip\_tol=1.5, box\_type='bb'):

self.max\_boxes = max\_boxes

self.min\_points\_in\_contour = min\_points\_in\_contour

self.min\_area = min\_area

self.min\_box\_aspect = min\_box\_aspect

self.angle\_tol = angle\_tol

self.lineskip\_tol = lineskip\_tol

self.box\_type = box\_type

def \_\_call\_\_(self, img\_binary):

cs = measure.find\_contours(img\_binary, 0.5)

# Collect contours into RotatedBoxes

results = []

for c in cs:

# Now examine the bounding box. If it is too small, we ignore the contour

ll, ur = np.min(c, 0), np.max(c, 0)

wh = ur - ll

if wh[0]\*wh[1] < self.min\_area: continue

# Finally, construct the rotatedbox. If its aspect ratio is too small, we ignore it

rb = RotatedBox.from\_points(c, self.box\_type)

if rb.height == 0 or rb.width/rb.height < self.min\_box\_aspect: continue

# All tests fine, add to the list

results.append(rb)

# Next sort and leave only max\_boxes largest boxes by area

results.sort(lambda x,y: 1 if x.area < y.area else -1)

return self.\_merge\_boxes(results[0:self.max\_boxes])

def \_are\_aligned\_angles(self, b1, b2):

"Are two boxes aligned according to their angle?"

return abs(b1 - b2) <= self.angle\_tol or abs(np.pi - abs(b1 - b2)) <= self.angle\_tol

def \_are\_nearby\_parallel\_boxes(self, b1, b2):

"Are two boxes nearby, parallel, and similar in width?"

if not self.\_are\_aligned\_angles(b1.angle, b2.angle): return False

# Otherwise pick the smaller angle and see whether the two boxes are close according to the "up" direction wrt that angle

angle = min(b1.angle, b2.angle)

return abs(np.dot(b1.center - b2.center, [-np.sin(angle), np.cos(angle)])) < self.lineskip\_tol\*(b1.height + b2.height) and \

(b1.width > 0) and (b2.width > 0) and (0.5 < b1.width/b2.width < 2.0)

def \_merge\_any\_two\_boxes(self, box\_list):

"""Given a list of boxes, finds two nearby parallel ones and merges them. Returns false if none found."""

for i in range(len(box\_list)):

for j in range(i+1,len(box\_list)):

if self.\_are\_nearby\_parallel\_boxes(box\_list[i], box\_list[j]):

# Remove the two boxes from the list, add a new one

a, b = box\_list[i], box\_list[j]

merged\_points = np.vstack([a.points, b.points])

merged\_box = RotatedBox.from\_points(merged\_points, self.box\_type)

if merged\_box.width/merged\_box.height >= self.min\_box\_aspect:

box\_list.remove(a)

box\_list.remove(b)

box\_list.append(merged\_box)

return True

return False

def \_merge\_boxes(self, box\_list):

"""Mergest nearby parallel boxes in the given list."""

while self.\_merge\_any\_two\_boxes(box\_list):

pass

return box\_list

class FindFirstValidMRZ(object):

"""Iterates over boxes found by MRZBoxLocator, passes them to BoxToMRZ, finds the first valid MRZ

or the best-scoring MRZ"""

\_\_provides\_\_ = ['box\_idx', 'roi', 'text', 'mrz']

\_\_depends\_\_ = ['boxes', 'img', 'img\_small', 'scale\_factor', '\_\_data\_\_']

def \_\_init\_\_(self, use\_original\_image=True):

self.box\_to\_mrz = BoxToMRZ(use\_original\_image)

def \_\_call\_\_(self, boxes, img, img\_small, scale\_factor, data):

mrzs = []

data['\_\_debug\_\_mrz'] = []

for i, b in enumerate(boxes):

roi, text, mrz = self.box\_to\_mrz(b, img, img\_small, scale\_factor)

data['\_\_debug\_\_mrz'].append((roi, text, mrz))

if mrz.valid:

return i, roi, text, mrz

elif mrz.valid\_score > 0:

mrzs.append((i, roi, text, mrz))

if len(mrzs) == 0:

return None, None, None, None

else:

mrzs.sort(cmp = lambda x,y: x[3].valid\_score - y[3].valid\_score)

return mrzs[-1]

class BoxToMRZ(object):

"""Extracts ROI from the image, corresponding to a box found by MRZBoxLocator, does OCR and MRZ parsing on this region."""

\_\_provides\_\_ = ['roi', 'text', 'mrz']

\_\_depends\_\_ = ['box', 'img', 'img\_small', 'scale\_factor']

def \_\_init\_\_(self, use\_original\_image=True):

"""

:param use\_original\_image: when True, the ROI is extracted from img, otherwise from img\_small

"""

self.use\_original\_image = use\_original\_image

def \_\_call\_\_(self, box, img, img\_small, scale\_factor):

img = img if self.use\_original\_image else img\_small

scale = 1.0/scale\_factor if self.use\_original\_image else 1.0

# If the box's angle is np.pi/2 +- 0.01, we shall round it to np.pi/2:

# this way image extraction is fast and introduces no distortions.

# and this may be more important than being perfectly straight

# similar for 0 angle

if abs(abs(box.angle) - np.pi/2) <= 0.01:

box.angle = np.pi/2

if abs(box.angle) <= 0.01:

box.angle = 0.0

roi = box.extract\_from\_image(img, scale)

text = ocr(roi)

if '>>' in text or ('>' in text and '<' not in text):

# Most probably we need to reverse the ROI

roi = roi[::-1,::-1]

text = ocr(roi)

if not '<' in text:

# Assume this is unrecoverable and stop here (TODO: this may be premature, although it saves time on useless stuff)

return roi, text, MRZ.from\_ocr(text)

mrz = MRZ.from\_ocr(text)

mrz.aux['method'] = 'direct'

# Now try improving the result via hacks

if not mrz.valid:

text, mrz = self.\_try\_larger\_image(roi, text, mrz)

# Sometimes the filter used for enlargement is important!

if not mrz.valid:

text, mrz = self.\_try\_larger\_image(roi, text, mrz, 1)

if not mrz.valid:

text, mrz = self.\_try\_black\_tophat(roi, text, mrz)

return roi, text, mrz

def \_try\_larger\_image(self, roi, cur\_text, cur\_mrz, filter\_order=3):

"""Attempts to improve the OCR result by scaling the image. If the new mrz is better, returns it, otherwise returns

the old mrz."""

if roi.shape[1] <= 700:

scale\_by = int(1050.0/roi.shape[1] + 0.5)

roi\_lg = transform.rescale(roi, scale\_by, order=filter\_order)

new\_text = ocr(roi\_lg)

new\_mrz = MRZ.from\_ocr(new\_text)

new\_mrz.aux['method'] = 'rescaled(%d)' % filter\_order

if new\_mrz.valid\_score > cur\_mrz.valid\_score:

cur\_mrz = new\_mrz

cur\_text = new\_text

return cur\_text, cur\_mrz

def \_try\_black\_tophat(self, roi, cur\_text, cur\_mrz):

roi\_b = morphology.black\_tophat(roi, morphology.disk(5))

new\_text = ocr(roi\_b) # There are some examples where this line basically hangs for an undetermined amount of time.

new\_mrz = MRZ.from\_ocr(new\_text)

if new\_mrz.valid\_score > cur\_mrz.valid\_score:

new\_mrz.aux['method'] = 'black\_tophat'

cur\_text, cur\_mrz = new\_text, new\_mrz

new\_text, new\_mrz = self.\_try\_larger\_image(roi\_b, cur\_text, cur\_mrz)

if new\_mrz.valid\_score > cur\_mrz.valid\_score:

new\_mrz.aux['method'] = 'black\_tophat(rescaled(3))'

cur\_text, cur\_mrz = new\_text, new\_mrz

return cur\_text, cur\_mrz

class TryOtherMaxWidth(object):

"""

If mrz was not found so far in the current pipeline,

changes the max\_width parameter of the scaler to 1000 and reruns the pipeline again.

"""

\_\_provides\_\_ = ['mrz\_final']

\_\_depends\_\_ = ['mrz', '\_\_pipeline\_\_']

def \_\_init\_\_(self, other\_max\_width=1000):

self.other\_max\_width = other\_max\_width

def \_\_call\_\_(self, mrz, \_\_pipeline\_\_):

# We'll only try this if we see that img\_binary.mean() is very small or img.mean() is very large (i.e. image is mostly white).

if mrz is None and (\_\_pipeline\_\_['img\_binary'].mean() < 0.01 or \_\_pipeline\_\_['img'].mean() > 0.95):

\_\_pipeline\_\_.replace\_component('scaler', Scaler(self.other\_max\_width))

new\_mrz = \_\_pipeline\_\_['mrz']

new\_mrz.aux['method'] = new\_mrz.aux['method'] + '|max\_width(%d)' % self.other\_max\_width

mrz = new\_mrz

return mrz

class MRZPipeline(Pipeline):

"""This is the "currently best-performing" pipeline for parsing MRZ from a given image file."""

def \_\_init\_\_(self, filename):

super(MRZPipeline, self).\_\_init\_\_()

self.version = '1.0' # In principle we might have different pipelines in use, so possible backward compatibility is an issue

self.filename = filename

self.add\_component('loader', Loader(filename))

self.add\_component('scaler', Scaler())

self.add\_component('boone', BooneTransform())

self.add\_component('box\_locator', MRZBoxLocator())

self.add\_component('mrz', FindFirstValidMRZ())

self.add\_component('other\_max\_width', TryOtherMaxWidth())

@property

def result(self):

return self['mrz\_final']

def read\_mrz(filename, save\_roi=False):

"""The main interface function to this module, encapsulating the recognition pipeline.

Given an image filename, runs MRZPipeline on it, returning the parsed MRZ object.

:param save\_roi: when this is True, the .aux['roi'] field will contain the Region of Interest where the MRZ was parsed from.

"""

p = MRZPipeline(filename)

mrz = p.result

if mrz is not None:

if save\_roi: mrz.aux['roi'] = p['roi']

return mrz

################################ END CODE ########################################