

Projet Imagerie Numérique

Sujet n°2 - lecture automatique d'un texte dans l'image d'un document

Introduction

Dans le cadre de la polycompétence imagerie numérique, nous avons réalisé un projet qui consiste à extraire un texte d'une image. Pour cela nous avons mené une étude bibliographique sur les librairies de python spécialisées dans la reconnaissance optique de caractère (OCR) et la vision par ordinateur. Nous avons relevé deux librairies intéressantes que nous vous présenterons. Nous nous sommes ensuite renseigné sur les librairies qui possèdent des fonctions qui puissent redresser une image, car nous avons en effet besoin de redresser et recadrer automatiquement l'image pour rendre son texte plus lisible. Une fois cette étape réalisée nous avons implémenté une fonction pour extraire le texte, selon la librairie que nous avons choisie selon l'étude bibliographique.

Nous vous présenterons notre code détaillé en revenant sur les fonctions les plus importantes.

Etude bibliographique sur les librairies de reconnaissance optique de caractère

Lorsque nous avons effectué des recherches sur les librairies de reconnaissance optique des caractères, deux nous ont semblé particulièrement pertinentes pour mener notre projet : Tesseract et EasyOCR. Nous vous présenterons ces deux outils indépendamment, en vous expliquant leur fonctionnement grâce à des exemples. Dans la partie où nous présenterons Tesseract, nous ferons également un point sur la librairie OpenCV, qui nous a servi à implémenter notre projet.

OpenCV et Tesseract

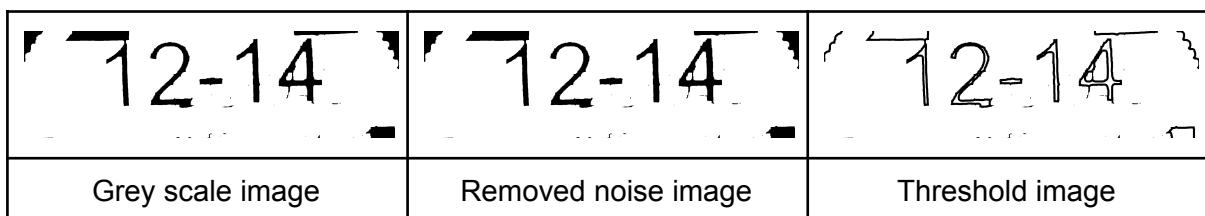
OpenCV, acronyme de Open Source Computer Vision, est une librairie graphique libre de droit spécialisée dans le traitement d'image et la vision par ordinateur. Elle permet plusieurs opérations classiques telles que la lecture, l'écriture et l'affichage d'une image, calcul de l'histogramme d'une image, l'application de filtre, le seuillage ou encore la segmentation. Dans notre projet de lecture automatique d'un texte à partir d'une image nous allons utiliser la librairie **OpenCV** et l'outil open source OCR **Tesseract**. Ils sont tous les deux compatibles avec le langage Python.

C'est grâce à Tesseract que nous pourrons extraire le texte d'une image, et en particulier grâce à sa fonction `image_to_string`.

La lecture d'un texte à partir d'une image se décompose en plusieurs étapes :

- Prétraitement de l'image
- Localisation de texte
- Reconnaissance des caractères/mots/lignes

Le prétraitement de l'image se fait grâce à OpenCV. Il s'agit de d'abord passer l'image en niveau de gris, puis enlever le bruit afin que la reconnaissance soit plus efficace. On applique souvent un seuil (thresholding) pour avoir une image en noir et blanc. Une fois ces opérations faites nous avons une meilleure reconnaissance des caractères.



Ensuite, nous pouvons visualiser la localisation du texte/des caractères grâce à la fonction `image_to_boxes` (*Figures 1 et 2*).

```
import cv2
import pytesseract

pytesseract.pytesseract.tesseract_cmd = r'C:\Program Files\Tesseract-OCR\tesseract.exe'

img = cv2.imread('image2.png')

h, w, c = img.shape
boxes = pytesseract.image_to_boxes(img)
for b in boxes.splitlines():
    b = b.split(' ')
    img = cv2.rectangle(img, (int(b[1]), h - int(b[2])), (int(b[3]), h - int(b[4])), (0, 255, 0), 2)

cv2.imshow('img', img)
cv2.waitKey(0)
```

ETAPE 2 :

ON RAJOUTE DU TEXTE ET DE
LA PONCTUATION.
IL FAUT SEPARER LES LIGNES
PUIS LES MOTS PUIS LES
CARACTERES !

Figure 1 : Implémentation & résultat de la localisation des caractères

```
import cv2
from pytesseract import Output

pytesseract.pytesseract.tesseract_cmd = r'C:\Program Files\Tesseract-OCR\tesseract.exe'

img = cv2.imread('image2.png')

d = pytesseract.image_to_data(img, output_type=Output.DICT)
print(d['conf'][i])
n_boxes = len(d['text'])
for i in range(n_boxes):
    if int(float(d['conf'][i])) > 60:
        (x, y, w, h) = (d['left'][i], d['top'][i], d['width'][i], d['height'][i])
        img = cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 0), 2)

cv2.imshow('img', img)
cv2.waitKey(0)
```

ETAPE 2 :

**ON RAJOUTE DU TEXTE ET DE
LA PONCTUATION.**
**IL FAUT SEPARER LES LIGNES
PUIS LES MOTS PUIS LES
CARACTERES !**

Figure 2 : Implémentation & résultat de la détection des mots

Enfin, pour extraire le texte de l'image, la reconnaissance des caractères, des mots et des lignes se fait avec la fonction **image_to_string**. Voici un exemple de code qui extrait le texte d'une image (ici l'image est bien centrée) en passant par le pré-traitement (*Figure 3*).

```

def get_string(img_path):
    # Read image with opencv
    img = cv2.imread(img_path)

    # Convert to gray
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # Apply dilation and erosion to remove some noise
    kernel = np.ones((1, 1), np.uint8)
    img = cv2.dilate(img, kernel, iterations=1)
    img = cv2.erode(img, kernel, iterations=1)

    # Write image after removed noise
    cv2.imwrite("removed_noise.png", img)

    # Apply threshold to get image with only black and white
    img = cv2.adaptiveThreshold(img, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 31, 2)

    # Write the image after apply opencv to do some ...
    cv2.imwrite("thres.png", img)

    # Recognize text with tesseract for python
    result = pytesseract.image_to_string(Image.open("thres.png"))

    # Remove template file
    #os.remove(temp)

    return result

print('--- Start recognize text from image ---')
print(get_string("test_text.png"))

print("----- Done -----")

```

Figure 3 : Extraction du texte par Tesseract

On obtient alors le résultat ci-dessous (*Figure 4*).

ETAPE 2 : ON RAJOUTE DU TEXTE ET DE LA PONCTUATION. IL FAUT SEPARER LES LIGNES PUIS LES MOTS PUIS LES CARACTERES !	--- Start recognize text from image --- ETAPE 2: ON RAJOUTE DU TEXTE ET DE LA PONCTUATION. IL FAUT SEPARER LES LIGNES PUIS LES MOTS PUIS LES CARACTERES ! ----- Done -----
input	output

Figure 4 : Résultat de l'extraction de texte

Bien que Tesseract soit une des bibliothèques les plus populaires dans la reconnaissance de caractères, elle possède tout de même quelques défauts. Voici une liste de ses avantages et de ses inconvénients (*Figure 5*).

Avantages	Inconvénients
Flexible et puissant	Ne reconnaît pas d'écriture à la main
Open source	Ne marche pas très bien quand les images sont modifiées (flous, occlusions partielles, distorsion ...)
Bonne documentation	Analyse toujours les textes de gauche à droite (ne prend pas en compte d'éventuelles colonnes etc ...)
Moteur basé sur un réseau de neurones	Faible résultat avec des photos de mauvaise qualité
116 langages supportés	Peut parfois se tromper et renvoyer n'importe quoi
Autorise plusieurs mode de segmentation de texte	

Figure 5 : Avantages et Inconvénients Tesseract

EasyOCR

Dans cette partie, nous allons expliquer la deuxième Library EasyOCR qui nous permet d'extraire un texte d'une image.

Le lecteur optique de caractères ou l'OCR est une technique utilisée pour convertir le texte de documents visuels en texte codé. Ces éléments visuels peuvent être des documents imprimés (factures, relevés bancaires, notes de restaurant), des panneaux (panneaux de signalisation, symboles de circulation) ou des images.

EasyOCR est un paquetage python qui permet de convertir l'image en texte. C'est de loin la façon la plus simple de mettre en œuvre l'OCR et il a accès à plus de 80 langues, dont l'anglais et le français. Elle peut ainsi traiter plusieurs langues en même temps, à condition qu'elles soient compatibles entre elles.

EASYOCR est capable de reconnaître un texte sur une image à l'aide d'une simple fonction **recognize_text(imag_path)**.

- *Imag_path : Chemin de l'image*

```
1 def recognize_text(imag_path):  
2     reader = easyocr.Reader(['en'])  
3     return reader.readtext(imag_path)
```

Figure 6 : Fonction `recognize_text` EasyOCR

Reader permet d'initialiser le lecteur, il prend en argument une liste de codes de langue et d'autres paramètres. Reader peut traiter plusieurs langues en même temps. Si par exemple nous passons ['en'] en attribut, cela signifie qu'il ne détectera que la partie anglaise de l'image comme texte, s'il trouve d'autres langues comme le chinois ou le japonais, il ignorera ces textes.

Par exemple pour cette image :

危 机

DANGER

OPPORTUNITE

On fixe la langue que sur l'anglais, on obtient le résultat ci-dessous (Figure 7).

```
1 import matplotlib.pyplot as plt
2 import cv2
3 import easyocr
4 from pylab import rcParams
5 %matplotlib inline

1 def reconize_text(imag_path):
2     reader = easyocr.Reader(['en'])
3     return reader.readtext(imag_path,detail = 0)

1 resultat = reconize_text('test1.png')
CUDA not available - defaulting to CPU. Note: This module is much faster with a GPU.

1 resultat
['E #L', 'DANGER', 'OPPORTUNITE']
```

Figure 7 : Exemple fonctionnement reconnaissance texte EasyOCR - langue anglais

Si on fixe la langue sur l'Anglais et le Chinois on obtient le résultat suivant (Figure 8).

```
1 image = cv2.imread('test1.png')
2 plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
3
```

<matplotlib.image.AxesImage at 0x7fdc4a9bdd60>



```
1 def reconize_text(imag_path):
2     reader = easyocr.Reader(['ch_sim', 'en'])
3     return reader.readtext(imag_path, detail = 0)
```

```
1 resultat = reconize_text('test1.png')
```

CUDA not available - defaulting to CPU. Note: This module is much faster with a GPU.

```
1 resultat
```

```
['危 机', 'DANGER', 'OPPORTUNITE']
```

Figure 8 : Exemple fonctionnement reconnaissance texte EasyOCR - langue anglais et chinois

Ensuite, la fonction retourne la méthode **readtext** qui est la méthode principale de la classe Reader.

Cette fonction retourne le résultat suivant (*Figure 9*).

```
[[[[12, 0], [290, 0], [290, 118], [12, 118]], '危 机', 0.3501080717091032),
 ([[41, 145], [101, 145], [101, 163], [41, 163]], 'DANGER',
  0.9993263492057693),
 ([[175, 143], [277, 143], [277, 163], [175, 163]],
  'OPPORTUNITE',
  0.9956942908450411)]]
```

Figure 9 : Résultat méthode readtext

Le code couleur de la figure 9 signifie :

- *Les 4 coordonnées de la boîte englobante (x, y) du texte.*
- *Le texte identifié.*
- *Le score de confiance : ce score est le niveau de confiance, donc il va de 0 à 1. Quand la valeur est proche de 0 la méthode n'est pas vraiment sûre si la chaîne qu'il a détectée est entièrement exacte, et plus la valeur est proche de 1 plus la méthode est précise.*

Par exemple pour cette image :



J'apprends le chinois

Avec le code suivant (Figure 10)

```
1 import matplotlib.pyplot as plt
2 import cv2
3 import easyocr
4 from pylab import rcParams
5 %matplotlib inline

1 def recognize_text(imag_path):
2     reader = easyocr.Reader(['fr'])
3     return reader.readtext(imag_path)

1 resultat = recognize_text('test.png')
CUDA not available - defaulting to CPU. Note: This module is much faster with a GPU.

1 resultat
```

Figure 10 : Reconnaissance texte EasyOCR - langue française

On obtient le résultat suivant (Figure 11).

```
[([[[24, 0], [320, 0], [320, 66], [24, 66]], '#7 # iXiE', 0.13267364772475596,
 ([[38, 64], [82, 64], [82, 90], [38, 90]], 'Moi', 0.9997042626196786),
 ([[106, 62], [182, 62], [182, 90], [106, 90]], 'étudier', 0.9997647491138979),
 ([[202, 62], [280, 62], [280, 90], [202, 90]], 'chinois', 0.9999618816480752),
 ([[52, 120], [261, 120], [261, 154], [52, 154]],
 'J'apprends le chinois',
 0.8695637072842775)]]
```

Figure 11 : Résultat reconnaissance texte en français

Comme nous l'avons vu auparavant, easyOcr ne détecte que les langues initialisées dans la fonction *Reader*. Dans notre cas, le programme va nous détecter que les textes écrits en français. Pour cela, le score de confiance pour les mots qui ne sont pas en français est très faible, contrairement aux textes écrits en français, on constate que le score de confiance est très élevé.

Ce type de sortie peut être difficile, nous pouvons donc passer le paramètre de détail à 0 pour une sortie plus simple dans la fonction **Readtext**.

Pour le même exemple ci-dessus on obtient le résultat ci-dessous (*Figure 12*).

```
1 import matplotlib.pyplot as plt
2 import cv2
3 import easyocr
4 from pylab import rcParams
5 %matplotlib inline

1 def recognize_text(imag_path):
2     reader = easyocr.Reader(['fr'])
3     return reader.readtext(imag_path, detail = 0)

1 resultat = recognize_text('test.png')
CUDA not available - defaulting to CPU. Note: This module is much faster with a GPU.

1 resultat
['#7 # iXiE', 'Moi', 'étudier', 'chinois', 'Japprends le chinois']
```

Figure 12 : Implémentation reconnaissance texte detail = 0

Implémentation de la détection d'un texte sur une image

Le sujet de notre projet est d'extraire un texte d'une image. Pour cela notre code se décompose en 4 étapes :

- **Etape 1** : Premièrement on charge l'image, on la redimensionne et on la passe en noir et blanc pour pouvoir effectuer des opérations dessus.
- **Etape 2** : On extrait les contours du plus grand rectangle dans l'image.
- **Etape 3** : On découpe et on effectue une rotation sur l'image pour faire en sorte que les coins du plus grands rectangles soient les coins de l'image.
- **Etape 4** : On extrait le texte de l'image

Nous allons vous détailler chaque étape pour une meilleure compréhension de notre code.

Etape 1

Pour commencer, nous avons besoin d'importer les packages et l'image. Prenons l'exemple de l'image ci-dessous (*Figure 13*). Il y a beaucoup d'informations dans notre image dont nous n'avons pas besoin.

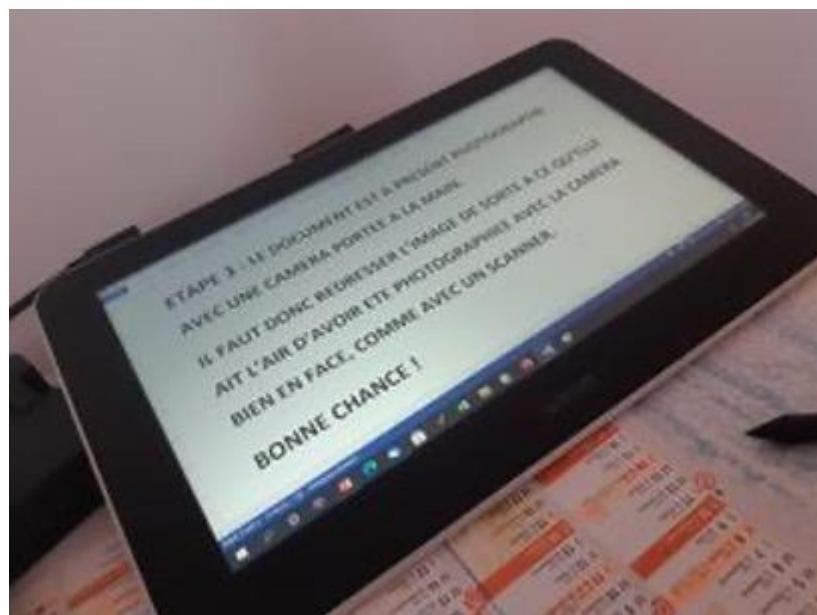


Figure 13 : Image de l'exemple

Nous n'avons pas besoin de l'information relative à la coque de la tablette. Et nous ne nous soucions certainement pas de l'arrière-plan sur lequel notre image a été photographiée. Tout ce qui nous intéresse, c'est l'écran et le texte qui est dessus. Car une fois que nous arrivons à garder juste le texte, nous pouvons procéder à l'identification du texte.

Dans cette partie et lors des prochaines étapes, nous allons vous expliquer comment trouver automatiquement un texte dans une image en utilisant uniquement Python et OpenCV. Plus précisément, nous allons utiliser la fonctionnalité de contours d'OpenCV et la fonction findContours du package cv2.

```
Entrée [1]: 1 import numpy as np
2 import cv2
3 from matplotlib import pyplot as plt
4 from mpl_toolkits.axes_grid1 import ImageGrid
5 import math
6 import imutils
7
8 img = cv2.imread('1.png')
9 plt.imshow(img)

Out[1]: <matplotlib.image.AxesImage at 0x7fe6db8165e0>


```

Figure 14 : Importation des packages et de l'image

Les lignes 1 à 6 de la figure 14 gèrent simplement l'importation des packages. Nous utiliserons skimage, NumPy, argparse pour analyser les arguments de la ligne de commande et cv2 contient nos liens OpenCV.

Etape 2

```
1 # load the query image, compute the ratio of the old height
2 # to the new height, clone it, and resize it
3 image = cv2.imread('1.png')
4 ratio = image.shape[0] / 300.0
5 orig = image.copy()
6 image = imutils.resize(image, height = 300)
7 # convert the image to grayscale, blur it, and find edges
8 # in the image
9 gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
10 gray = cv2.bilateralFilter(gray, 11, 17, 17)
11 edged = cv2.Canny(gray, 30, 200)
```

Figure 15 : Traitements sur l'image

Afin d'accélérer les étapes de traitement illustrés par la figure 15, nous devons redimensionner l'image. Plus l'image est petite, plus le traitement est rapide. En contrepartie, si nous réduisons trop la taille de l'image, nous perdons des détails importants.

Dans ce cas, nous voulons que la hauteur de notre nouvelle image soit de 300 pixels. À la ligne 4, nous calculons le rapport entre l'ancienne et la nouvelle hauteur, puis nous créons un clone de l'image originale à la ligne 5. Enfin, la ligne 6 gère le redimensionnement de l'image à une hauteur de 300 pixels.

Ensuite, nous convertissons notre image en niveaux de gris à la ligne 9. Nous floutons ensuite légèrement l'image en utilisant la fonction cv2.bilateralFilter.

Le filtrage bilatéral a l'avantage de supprimer le bruit dans l'image tout en préservant les bords réels. Les bords sont importants car nous en aurons besoin pour trouver le texte dans l'image.

Enfin, nous appliquons la détection des bords de Canny à la ligne 11.

```
1 # find contours in the edged image, keep only the largest
2 # ones, and initialize our screen contour
3 cnts = cv2.findContours(edged.copy(), cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
4 cnts = imutils.grab_contours(cnts)
5 cnts = sorted(cnts, key = cv2.contourArea, reverse = True)[:10]
6 screenCnt = None
```

Figure 16 : Recherche des contours de l'image

Afin de trouver l'écran de la tablette dans notre image avec bords, nous devons trouver les contours de l'image.

Pour trouver les contours d'une image, nous avons besoin de la fonction OpenCV cv2.findContours à la ligne 3 de la figure 16. Cette méthode requiert trois paramètres. Le premier est l'image dans laquelle nous voulons trouver des contours. Nous passons notre image avec des bords, en nous assurant de la cloner d'abord. La méthode cv2.findContours est destructive (c'est-à-dire qu'elle manipule l'image que vous lui transmettez), donc si vous prévoyez de réutiliser cette image plus tard, assurez-vous de la cloner. Le deuxième paramètre cv2.RETR_TREE indique à OpenCV de calculer la hiérarchie (relation) entre les

contours. Enfin, nous demandons à OpenCV de compresser les contours pour gagner de la place en utilisant cv2.CV_CHAIN_APPROX_SIMPLE.

En retour, la fonction cv2.findContours nous donne une liste des contours qu'elle a trouvés, mais nous devons l'analyser à la ligne 4 en raison de la façon dont les différentes versions d'OpenCV gèrent les contours.

La première chose à faire est de réduire le nombre de contours à traiter. Nous savons que la surface de l'écran de la tablette est assez grande par rapport au reste des régions de l'image. La ligne 5 permet de trier nos contours, du plus grand au plus petit, en calculant l'aire du contour à l'aide de cv2.contourArea. Nous n'avons maintenant que les 10 plus grands contours. Enfin, nous initialisons screenCnt, le contour qui correspond à notre écran à la ligne 6.

```
1 # loop over our contours
2 for c in cnts:
3     # approximate the contour
4     peri = cv2.arcLength(c, True)
5     approx = cv2.approxPolyDP(c, 0.015 * peri, True)
6     # if our approximated contour has four points, then
7     # we can assume that we have found our screen
8     if len(approx) == 4:
9         screenCnt = approx
10        break
```

Figure 17 : Recherche du contour correspondant à l'écran de la tablette

À la ligne 2 de la figure 17, nous commençons à boucler sur nos 10 plus grands contours dans l'image de la requête. Ensuite, nous approximons le contour en utilisant cv2.arcLength et cv2.approxPolyDP. Ces méthodes sont utilisées pour approximer les courbes polygonales d'un contour. Afin d'approximer un contour, vous devez fournir votre niveau de précision d'approximation. Dans ce cas, nous utilisons 1,5 % du périmètre du contour.

À la ligne 8, nous vérifions combien de points possède notre contour approximatif. Si le contour a quatre points, il s'agit (probablement) de notre écran de tablette. Si le contour à quatre points, nous stockons alors notre contour approximatif à la ligne 9.

Et ensuite, nous dessinons les contours (*Figure 18*).

```
1 cv2.drawContours(image, [screenCnt], -1, (0, 255, 0), 3)
2 cv2.imshow("Texte Screen", image)
3 cv2.waitKey(0)
```

Figure 18 : Tracé des contours de l'image

Et on obtient le résultat ci-dessous (*Figure 19*).

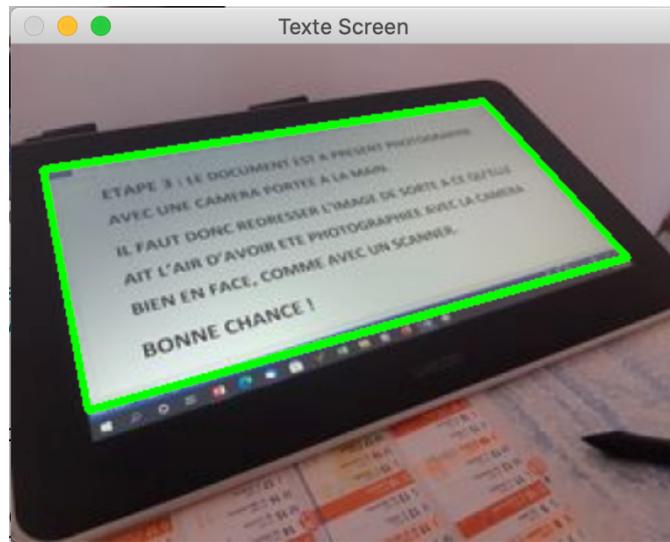


Figure 19 : Détection des contours du texte

Etape 3

Lors de cette étape, nous avons déjà les 4 coins correspondant au plus grand rectangle dans l'image. De là notre première étape sera de récupérer la largeur et la hauteur de l'image initiale (*Figure 20*)

```
import math
import cv2
import scipy.spatial.distance
import numpy as np

#rows est une variable contenant la hauteur de l'image, cols content sa largeur
(rows,cols,_) = image.shape

#On trouve le centre de l'image
u0 = (cols)/2.0
v0 = (rows)/2.0
```

Figure 20 : Récupération de la hauteur et de la largeur de l'image

On calcule les coordonnées du centre de l'image, u_0 et v_0 , qui vont nous servir par la suite. On crée ensuite une liste avec les 4 coins du plus grand rectangle (*Figure 21*).

```
#On initialise une liste qui contient les coordonnées des 4 coins du plus grand rectangle de l'image
#(trouvé à l'étape précédente)
#p[0] = coin supérieur gauche
#p[1] = coin supérieur droit
#p[2] = coin inférieur gauche
#p[3] = coin inférieur droit
p = []
p.append((screenCnt[2][0][0],screenCnt[2][0][1]))
p.append((screenCnt[1][0][0],screenCnt[1][0][1]))
p.append((screenCnt[3][0][0],screenCnt[3][0][1]))
p.append((screenCnt[0][0][0],screenCnt[0][0][1]))
```

Figure 21 : Création d'une liste possédant les coins du contour du rectangle

On va ensuite calculer la distance euclidienne entre les coins supérieurs droit et gauche, et les coins inférieurs droit et gauche (*Figure 22*). On va donc avoir deux nouvelles largeurs et deux nouvelles hauteurs pour la nouvelle image.

```
#On calcule la distance euclidienne entre les coins supérieurs gauche et droit, et entre les coins inférieur gauche et droit
#w1 et w2 correspondent à la largeur de la nouvelle image recadrée (w1 : largeur entre les coins supérieurs,
#w2 : largeur entre les coins inférieurs).
w1 = scipy.spatial.distance.euclidean(p[0],p[1])
w2 = scipy.spatial.distance.euclidean(p[2],p[3])

#de la même manière, on calcule la hauteur de la nouvelle image (h1 entre les coins à gauche, h2 entre les coins à droite)
h1 = scipy.spatial.distance.euclidean(p[0],p[2])
h2 = scipy.spatial.distance.euclidean(p[1],p[3])
```

Figure 22 : Calcul de la distance euclidienne entre les coins

La fonction `scipy.spatial.distance.euclidean` calcule la distance euclidienne entre les deux vecteurs. Elle applique cette formule :

$$\|u - v\|_2$$

Une fois les hauteurs et largeurs calculées, va prendre le maximum de ces hauteurs et ces largeurs comme la hauteur et la largeur de la nouvelle image, pour ne pas perdre d'information (*Figure 23*).

On va également calculer le ratio entre la largeur et la hauteur de la nouvelle image.

```
#Etant donné que nous avons deux hauteurs et deux largeurs, on prend la hauteur et la largeur maximales
#pour être sur de ne pas perdre d'information
w = max(w1,w2)
h = max(h1,h2)

#On calcule le ratio entre la largeur et la hauteur
ar_vis = float(w)/float(h)
```

Figure 23 : Récupération du maximum des hauteurs et largeurs

On convertit ensuite chaque coin du rectangle en type array grâce à la fonction `numpy.array` pour pouvoir plus facilement effectuer des opérations dessus (*Figure 24*). On caste les valeurs en type float. Il est en effet plus facile de faire des opérations mathématiques sur des array plutôt que des listes.

```
#On transforme les coins en numpy array pour pouvoir faire plus facilement des opérations dessus
m1 = np.array((p[0][0],p[0][1],1)).astype('float32')
m2 = np.array((p[1][0],p[1][1],1)).astype('float32')
m3 = np.array((p[2][0],p[2][1],1)).astype('float32')
m4 = np.array((p[3][0],p[3][1],1)).astype('float32')
```

Figure 24 : Transformation des coins en type array

On va ensuite calculer la distance focale de l'image pour pouvoir la redresser automatiquement (*Figure 25*).

```
#On calcule la distance focale

k2 = np.dot(np.cross(m1,m4),m3) / np.dot(np.cross(m2,m4),m3)
k3 = np.dot(np.cross(m1,m4),m2) / np.dot(np.cross(m3,m4),m2)

n2 = k2 * m2 - m1
n3 = k3 * m3 - m1

n21 = n2[0]
n22 = n2[1]
n23 = n2[2]

n31 = n3[0]
n32 = n3[1]
n33 = n3[2]

f = math.sqrt(np.abs( (1.0/(n23*n33)) * ((n21*n31 - (n21*n33 + n23*n31)*u0 + n23*n33*u0*u0) +
(n22*n32 - (n22*n33+n23*n32)*v0 + n23*n33*v0*v0)))))

A = np.array([[f,0,u0],[0,f,v0],[0,0,1]]).astype('float32')

At = np.transpose(A)
Ati = np.linalg.inv(At)
Ai = np.linalg.inv(A)
```

Figure 25 : Calcul de la distance focale

np.dot est une fonction permettant de faire un produit matriciel, et np.cross permet de faire un produit vectoriel.

A partir de là on va calculer le ratio réel entre la largeur et la hauteur, en fonction des matrices qu'on a trouvé précédemment.

On va comparer ce ratio avec le précédent ratio calculé, en fonction de ça on va pouvoir déterminer une hauteur et une largeur pour la nouvelle image (*Figure 26*).

```
#On calcule le réel ratio entre la largeur et la hauteur
ar_real = math.sqrt(np.dot(np.dot(np.dot(n2,Ati),Ai),n2)/np.dot(np.dot(np.dot(n3,Ati),Ai),n3))

#On initialise la hauteur et la largeur de la nouvelle image en fonction du ratio calculé précédemment
if ar_real < ar_vis:
    W = int(w)
    H = int(W / ar_real)
else:
    H = int(h)
    W = int(ar_real * H)
```

Figure 26 : Comparaison des ratios

On initialise ensuite deux variables pts1 et pts2 qui correspondent à l'ensemble des coins calculés lors de l'étape précédente et aux coins réels de la nouvelle image, avec sa nouvelle hauteur et largeur (*Figure 27*).

```
pts1 = np.array(p).astype('float32')
pts2 = np.float32([[0,0],[W,0],[0,H],[W,H]])
```

Figure 27 : Initialisation de pts1 et pts2

Ensuite, on va utiliser la fonction getPerspectiveTransform pour obtenir une matrice de transformation de l'image, en fonction des pts1 et pts2 (donc des coins de l'ancienne image, et des coins de la nouvelle). Cette matrice, une fois appliquée à l'image, va transformer les anciens coins en les nouveaux coins. (Voir figure 28)

C'est le rôle de la fonction warpPerspective, elle applique la matrice de transformation à l'image suivant des certaines dimensions. La sortie de cette fonction est la nouvelle image recadrée.

```
#La matrice M correspond à la matrice de transformation de l'image en fonction des nouveaux coins et des nouvelles mesures
M = cv2.getPerspectiveTransform(pts1,pts2)

#Cette fonction applique la matrice de transformation à l'image
#dst est notre image finale redressée et recadrée
dst = cv2.warpPerspective(image,M,(W,H))
```

Figure 28 : Transformation de l'image

On va ensuite afficher l'image originale, l'image recadrée et les enregistrer en tant que png et jpg (*Figure 29*).

```
cv2.imshow('img',image)
cv2.imshow('dst',dst)
cv2.imwrite('dst.jpg',dst)
cv2.imwrite('orig.png',image)
cv2.imwrite('proj.png',dst)

cv2.waitKey(0)
```

Figure 29 : Enregistrement de l'image recadrée

Suite à ces opérations, on obtient l'image redressée suivante (*Figure 30*).

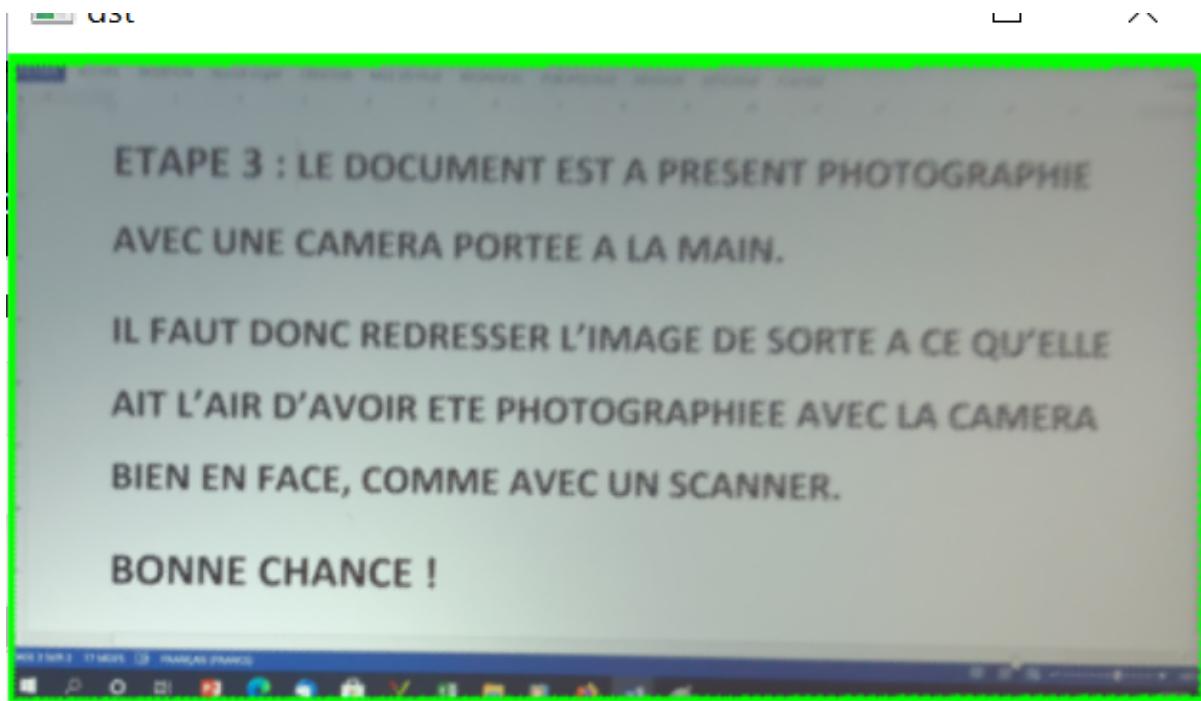


Figure 30 : Image redressée

On peut alors maintenant venir extraire le texte.

Etape 4

Pour extraire le texte de l'image, on a utilisé les deux méthodes documentées plus haut : avec Tesseract et EasyOCR.

Avec Tesseract

Une fois l'image recadrée et en face de la caméra, on pourra enfin déchiffrer le texte sur cette dernière.

Pour cela, on va utiliser les la librairie OpenCV, et l'outil de reconnaissance optique de caractère Tesseract présentés précédemment.

La première étape consiste à lire l'image avec OpenCV, via la ligne suivante (*Figure 31*).

```
# Lire l'image avec openCV
img = cv2.imread(img_path)
```

Figure 31 : Lecture de l'image

On effectue ensuite une série de traitement sur l'image pour la rendre plus lisible, cela à pour but d'augmenter le taux de bon déchiffrement du texte.

Dans l'ordre, on a (voir figure 32):

- convertit l'image en niveau de gris
- enlever le bruit en dilatant et érodant l'image
- appliquer un seuil pour avoir l'image en noir et blanc

```
# Convertir l'image en niveau de gris
img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Effectuer des traitements sur l'image pour enlever le bruit
kernel = np.ones((1, 1), np.uint8)
img = cv2.dilate(img, kernel, iterations=1)
img = cv2.erode(img, kernel, iterations=1)

# On applique une seuil pour avoir uniquement une image en noir et blanc
img = cv2.adaptiveThreshold(img, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 31, 2)
```

Figure 32 : Traitements sur l'image afin de la rendre plus lisible

Finalement, on applique à l'image la fonction de tesseract `image_to_string` pour extraire le texte de l'image (*Figure 33*).

```
# Reconnaissance du texte avec tesseract
result = pytesseract.image_to_string(Image.open("thres.png"))
```

Figure 33 : Fonction `image_to_string`

On permet ensuite à l'utilisateur d'enregistrer le texte issu de l'image dans un fichier au format txt. On obtient alors le texte suivant (*Figure 34*).

8 ee a mi TE rr,

ETAPE 3: LE DOCUMENT EST A PRESENT PHOTOGRAPHEE
AVEC UNE CAMERA PORTEE ALA MAIN,

iL FAUT DONC REDRESSER L'IMAGE DE SORTE A CE QU'ELEEE
AIT L'AIR D/AVOIR ETE PHOTOGRAPHIEE AVEC 4A CAMERA,
BIEN EN FACE, COMME AVEC UN SCANNER.

BONNE CHANCE I

Figure 34 : Texte extrait de l'image

Avec EasyOCR

On a un résultat similaire avec la librairie EasyOCR (*Figure 35*).

```
import matplotlib.pyplot as plt
import cv2
import easyocr
from pylab import rcParams
%matplotlib inline

def reconize_text(imag_path):
    reader = easyocr.Reader(['en'])
    return reader.readtext(imag_path)

resultat = reconize_text('dst.jpg')

print(resultat)
```

Figure 35 : Implémentation de l'extraction du texte avec EasyOCR

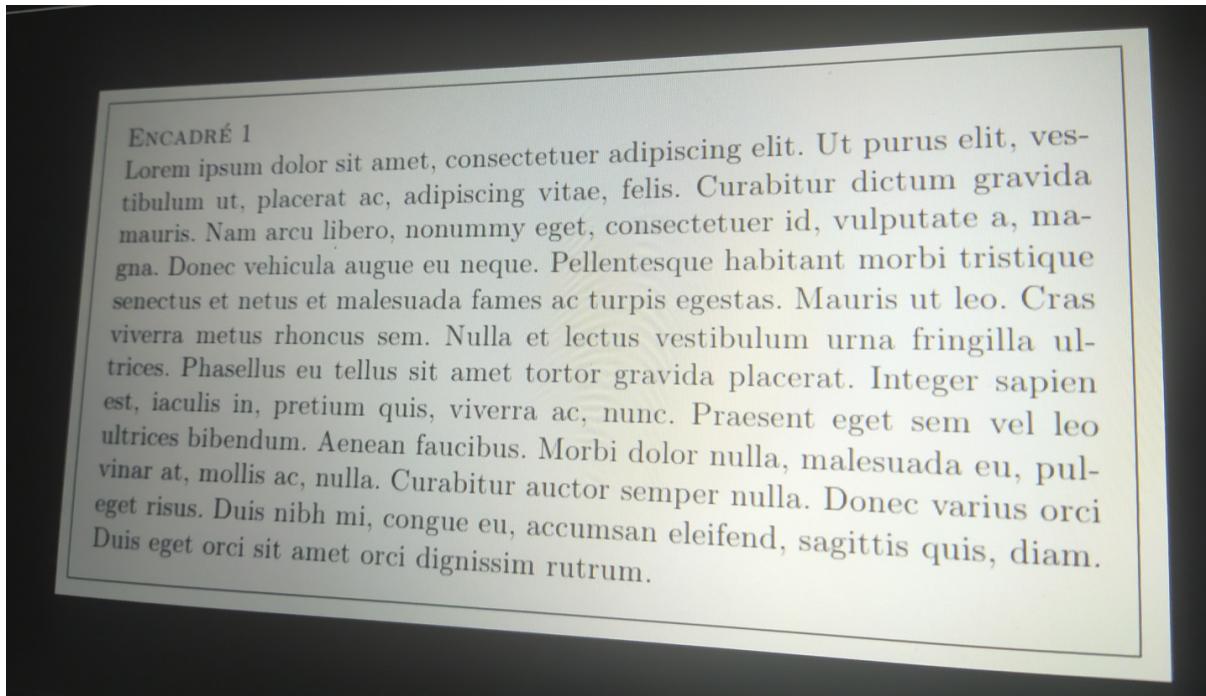
On obtient le résultat suivant (*Figure 36*).

```
[([[[35, 31], [79, 31], [79, 47], [35, 47]], 'ETAPE', 0.998675199430464), ([[91, 33], [213, 33], [213, 49], [91, 49]]], ':LE DOCUMENT EST', 0.6479945472091454), ([[225, 35], [379, 35], [379, 51], [225, 51]], 'PRESENT PHOTOGRAPHIE', 0.5071053536902849), ([[35, 57], [273, 57], [273, 77], [35, 77]], 'AVEC UNE CAMERA PorteE A LA Main:', 0.4029875169787284), ([[35, 89], [387, 89], [387, 109], [35, 109]]], "IL FAUT DONC REDRESSER L'IMAGE De Sorte A Ce QU'ELLE", 0.11505074623074092), ([[35, 115], [381, 115], [381, 135], [35, 135]]], "Ait L'AIR D'Avoir ETe PhotograPhiEE AVEC LA CAMERA", 0.24084507580864525), ([[35, 141], [291, 141], [291, 159], [35, 159]]], 'BIEN EN FACE, COMME AVEC UN SCANNER:', 0.799830633699558), ([[35, 173], [145, 173], [145, 189], [35, 189]], 'BONNE CHANCE', 0.993162006774032)]]
```

Figure 36 : Résultat avec EasyOCR

Résultat finaux

Le but de ce projet est de déchiffrer un texte dans n'importe quelle image. Pour qu'il ait un sens, nous l'avons testé avec des photos prises avec nos appareils personnels.
Prenons l'exemple de la photo ci-dessous.



Le programme extrait le texte suivant :

Encapaé 1 foals an Sh gh oy
Lorem ipsum dolor sit amet, consectetur adi . Ut purus elit, ves-
tibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida
mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, ma-
gna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique
senectus et netus et malesuada fames sc turpis egestas. Mauris ut leo. Cras
viverra metus rhoncus sem. Nulla et lectus vestibulum urna friny
trices. Phasellus eu tellus sit amet torto' ida placerat. Integer sapien
est, iaculis in, pretium quis, viverra ic. Praesent. eget sem vel leo
ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pul-
vinar at, mollis ac, nulla, Curabitur auctor semper nulla, Donec varius orci
eget risus, Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam,
Duis eget orci sit amet orci dignissim rutrum.

Conclusion

Notre code est fonctionnel, il fonctionne sur les images données à titre d'exemple lors des séances de projet, et sur nos photos personnels. Cependant, il n'a pas un taux de bonnes réponses de 100%, il se trompe sur certains mots, surtout quand la qualité de l'image est mauvaise et quand il y a de la ponctuation.

Finalement, nous avons essayé les deux outils OCR Tesseract et EasyOCR. EasyOCR semble légèrement meilleur que Tesseract, bien qu'il ne reconnaisse pas certains signes de ponctuation et certaines prépositions.

Sources

<https://www.opcito.com/blogs/extracting-text-from-images-with-tesseract-ocr-opencv-and-python>
<https://www.datacorner.fr/tesseract-adv/>
<https://nanonets.com/blog/ocr-with-tesseract/>
<https://bigdatablog.skapane.com/5-min-pour-comprendre-locr-partie-2>