

Comaprison of SOTA OCR tools

Week 2



February 5, 2024

Divyanshu

Image Preprocessing for OCR Tools

# Introduction

## Background

Preprocessing is crucial for Optical Character Recognition (OCR) tools to achieve enhanced outcomes because it helps to improve the quality of the input images, making it easier for the OCR engine to accurately recognize and extract text. Preprocessing is necessary for several reasons:

* Noise Reduction:
* Contrast Enhancement
* Image Binarization
* Rotation and Skew Correction
* Resizing and Scaling
* Text Line Segmentation

# Tools and Libraries

## OpenCV

* Open-source computer vision and image processing library.
* Supports tasks like image analysis, object recognition, and feature extraction.
* Widely used in computer vision applications, robotics, machine learning, and augmented reality.

## Pillow

* Python Imaging Library (PIL) Fork with added features.
* Used for basic image processing operations in Python.
* Complements OpenCV by providing a more Pythonic interface and supporting additional image formats.

## Pytesseract

* Python wrapper for Google's Tesseract OCR Engine.
* Enables text extraction from images.
* Integrates with OpenCV and Pillow for comprehensive image preprocessing and manipulation before OCR.

# Image Preprocessing Steps

## Image Loading

import cv2  
from PIL import Image  
import pytesseract  
import numpy as np  
  
im\_file = "data/Image1/download.jpeg"  
img = cv2.imread(im\_file)

## Inversion

Inverting the image enhances contrast and text visibility by reversing pixel values.

inverted\_image = cv2.bitwise\_not(img)  
cv2.imwrite("data/Image1/inverted.jpeg", inverted\_image)  
  
  
im = Image.open("data/Image1/inverted.jpeg")  
im.show()

## Grayscale Conversion

Converting the image to grayscale simplifies processing and reduces computational load.

def grayscale(img):  
 return cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)  
  
  
gray\_image = grayscale(fixed)  
cv2.imwrite("data/Image1/grayscale.jpeg", gray\_image)  
  
im = Image.open("data/Image1/grayscale.jpeg")  
im.show()

## Fixing Rotation

Fixing image rotation ensures proper orientation for accurate text recognition.

def getSkewAngle(cvImage) -> float:  
 # Prep image, copy, convert to gray scale, blur, and threshold  
 newImage = cvImage.copy()  
 gray = cv2.cvtColor(newImage, cv2.COLOR\_BGR2GRAY)  
 blur = cv2.GaussianBlur(gray, (9, 9), 0)  
 thresh = cv2.threshold(blur, 0, 255, cv2.THRESH\_BINARY\_INV + cv2.THRESH\_OTSU)[1]  
  
 kernel = cv2.getStructuringElement(cv2.MORPH\_RECT, (30, 5))  
 dilate = cv2.dilate(thresh, kernel, iterations=5)  
  
 # Find all contours  
 contours, hierarchy = cv2.findContours(dilate, cv2.RETR\_LIST, cv2.CHAIN\_APPROX\_SIMPLE)  
 contours = sorted(contours, key=cv2.contourArea, reverse=True)  
 for c in contours:  
 rect = cv2.boundingRect(c)  
 x, y, w, h = rect  
 cv2.rectangle(newImage, (x, y), (x + w, y + h), (0, 255, 0), 2)  
  
 largestContour = contours[0]  
 print(len(contours))  
 minAreaRect = cv2.minAreaRect(largestContour)  
 cv2.imwrite("data/Image1/boxes.jpg", newImage)  
  
 # Determine the angle. Convert it to the value that was originally used to obtain skewed image  
 angle = minAreaRect[-1]  
 if angle < -45:  
 angle = 90 + angle  
 return -1.0 \* angle  
  
  
def rotateImage(cvImage, angle: float):  
 newImage = cvImage.copy()  
 (h, w) = cvImage.shape[:2]  
 center = (w // 2, h // 2)  
 M = cv2.getRotationMatrix2D(center, angle, 1.0)  
 newImage = cv2.warpAffine(newImage, M, (w, h), flags=cv2.INTER\_CUBIC, borderMode=cv2.BORDER\_REPLICATE)  
 return newImage  
  
  
def deskew(cvImage):  
 angle = getSkewAngle(cvImage)  
 return rotateImage(cvImage, -1.0 \* angle)  
  
  
fixed = deskew(img)  
cv2.imwrite("data/Image1/rotation\_fixed.jpg", fixed)  
  
  
im = Image.open("data/Image1/rotation\_fixed.jpg")  
im.show()

## Binarization

Binarization simplifies the image by converting it into a binary form.

thresh, im\_bw = cv2.threshold(gray\_image, 165, 250, cv2.THRESH\_BINARY)  
cv2.imwrite("data/Image1/bw\_image.jpeg", im\_bw)  
  
  
im = Image.open("data/Image1/bw\_image.jpeg")  
im.show()

## Noise Reduction

Noise reduction improves image clarity by smoothing out irregularities.

def noise\_removal(image):  
 kernel = np.ones((1, 1), np.uint8)  
 image = cv2.dilate(image, kernel, iterations=1)  
 kernel = np.ones((1, 1), np.uint8)  
 image = cv2.erode(image, kernel, iterations=1)  
 image = cv2.morphologyEx(image, cv2.MORPH\_CLOSE, kernel)  
 image = cv2.medianBlur(image, 3)  
 return image  
  
  
no\_noise = noise\_removal(im\_bw)  
cv2.imwrite("data/Image1/no\_noise.jpeg", no\_noise)  
  
  
im = Image.open("data/Image1/no\_noise.jpeg")  
im.show()

## Dilation

Dilation enhances text features by expanding pixel values in the image.

def thick\_font(image):  
 image = cv2.bitwise\_not(image)  
 kernel = np.ones((2, 2), np.uint8)  
 image = cv2.dilate(image, kernel, iterations=1)  
 image = cv2.bitwise\_not(image)  
 return image  
  
  
dilated\_image = thick\_font(no\_noise)  
cv2.imwrite("data/Image1/dilated\_image.jpeg", dilated\_image)  
  
  
im = Image.open("data/Image1/dilated\_image.jpeg")  
im.show()

## Erosion

Erosion refines text regions by removing small, unwanted details.

def thin\_font(image):  
 image = cv2.bitwise\_not(image)  
 kernel = np.ones((2, 2), np.uint8)  
 image = cv2.erode(image, kernel, iterations=1)  
 image = cv2.bitwise\_not(image)  
 return image  
  
  
eroded\_image = thin\_font(no\_noise)  
cv2.imwrite("data/Image1/erroded\_image.jpeg", eroded\_image)  
  
  
im = Image.open("data/Image1/erroded\_image.jpeg")  
im.show()

## Removing Border

Removing borders eliminates unnecessary image edges, focusing on the central content.

def remove\_border(image):  
 contours, heirarchy = cv2.findContours(image, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)  
 cntSorted = sorted(contours, key=lambda x: cv2.contourArea(x))  
 cnt = cntSorted[-1]  
 x, y, w, h = cv2.boundingRect(cnt)  
 crop = image[y:y + h, x:x + w]  
 return crop  
  
  
no\_border = remove\_border(no\_noise)  
cv2.imwrite("data/Image1/no\_border.jpg", no\_border)  
  
im = Image.open("data/no\_border.jpg")  
im.show()  
  
color = [255, 255, 255]  
top, bottom, left, right = [150] \* 4  
  
image\_with\_border = cv2.copyMakeBorder(im, top, bottom, left, right, cv2.BORDER\_CONSTANT, value=color)  
cv2.imwrite("data/Image2/with\_border.jpg", image\_with\_border)  
  
im = Image.open("data/Image1/with\_border.jpg")  
im.show()

## Text Extraction

Images to Text

text = pytesseract.image\_to\_string(no\_border)  
# Print the extracted text  
print(text)

# Result and Evaluation

## Output Comparison

|  |  |
| --- | --- |
| Without Pre-processing | After Pre-processing |
| Nutrition Facts Servirig Size 1 Tbsp (15 mL) Servings Per Container About 64 Amount Per Serving Calories 0 Calories from Fato % Daily Value\* Total Fat 0g 0% Saturated Fat Og 0% Trans Fat 0g INGREDIENT Cholesterol Omg 0% APPLE CIDER Win Sodium Omg DISTRIBUTED § 0% | uous Potassium 15mg 0% | AUSTIN, th Total Carbohydrate Og ©2007 We 0% WWW. Wholeon VitaminA0% Vitamin C 0% Dietary Fiber 0g 0% CERTINED Oem Sugars 0g QUALITY A Protein 0g WHOLE fo FORMULATE INGREDIEW? Calcium 0% ° Iron 0% GENETICALLY PRODUCTS LAF Image ID: BRYGAS www.alamy.com | Nutrition Facts Serving Size 1 Tbsp (15 mL) Servings Per Container About 64  Amount Per Serving Calories 0 Calories from Fat 0  % Daily Vaiue\*  Total Fat 0g 0% Saturated Fat Og 0% | Trans Fat 0g  Cholesterol Omg 0%  Sodium 0mg 0% |  Potassium 15mg 0%  Total Carbohydrate 0g 0% Dietary Fiber 0g 0% Sugars Og  Protein 0g  VitaminA0% = VitaminC 0% Calcium0% = Iron 0% |
| **Nutrition Facts**  **Blueberries**  **lesterol 0 mg**  **\_Sodium 4 mg**  “Ca bohydrate 14g |  |

# Conclusion

In conclusion, image preprocessing plays a pivotal role in optimizing the performance of Tesseract OCR for text extraction. The combination of inversion, grayscale conversion, rotation fixing, dilation, erosion, noise reduction, and binarization collectively contributes to improving the clarity and interpretability of text within images.

When executed correctly, preprocessing not only enhances text recognition accuracy but also proves to be especially advantageous for nutrition label extraction. The preprocessing steps are particularly beneficial for table detection, ensuring a more precise and reliable output.