**TEXT DETECTION + DICTIONARY APPLICATION**

import cv2

import pytesseract

import matplotlib.pyplot as plt

import json

import numpy as np

import matplotlib.image as mpimg

from difflib import get\_close\_matches

from googletrans import Translator

image1 = cv2.imread("C:/Users/bandl/Deskewed Img.jpg")

img = cv2.resize(image1, (400, 400))

y=110

x=160

h=260

w=250

crop\_img = img[x:h, y:w]

cv2.imshow("cropped", crop\_img)

cv2.waitKey(0)

image\_bound = cv2.rectangle(img, (y,x), (w,h), (0,255,255), 2)

plt.figure(1)

plt.imshow(image1)

plt.title("Original Image Before resizing")

plt.show()

plt.figure(2)

plt.imshow(image\_bound)

plt.title("Image after resizing with bounding box")

plt.show()

plt.figure(3)

plt.imshow(crop\_img)

plt.title("Image extracted from Bounding Box")

plt.show()

cv2.imwrite("Cropped Image.jpg", crop\_img)

image\_thres=cv2.imread("C:/Users/bandl/Cropped Image.jpg")

crop\_img.shape

def rgb2gray(rgb):

r, g, b = rgb[:,:,0], rgb[:,:,1], rgb[:,:,2]

gray = 0.2989 \* r + 0.5870 \* g + 0.1140 \* b

return gray

img\_gray=rgb2gray(image\_thres)

img\_temp = np.ones((60,120), dtype=np.int16)

img\_gray=img\_temp\*(img\_gray > 120)

plt.figure(5)

plt.imshow(img\_gray,cmap='gray')

plt.title("Binarized Image")

pytesseract.pytesseract.tesseract\_cmd='C:/Users/bandl/Desktop/Tesseract-OCR/tesseract.exe'

text = pytesseract.image\_to\_string(img\_gray) #preprocessed AHE Image is called

print(text)

cv2.waitKey(0)

cv2.destroyAllWindows()

split\_sentence = text.split(' ')

#split\_sentence = map(lambda s: split\_sentence.strip(), split\_sentence)

m = len(split\_sentence)

x = split\_sentence[m-1].replace("\n", "")

split\_sentence[m-1] = x

print(split\_sentence)

translator = Translator()

#load JSON data

data = json.load(open("C:/Users/bandl/Desktop/data.json"))

def getMeaning(w):

w=w.lower()

#return data[w]

if w in data:

return data[w], w

#give matching word

elif len(get\_close\_matches(w,data.keys())) > 0:

close\_match = get\_close\_matches(w,data.keys())[0]

print("Did you mean '%s' instead? Enter Y if yes or N if no: " % close\_match)

choice = input()

choice = choice.lower()

if choice == 'y':

return data[close\_match], close\_match

elif choice == 'n':

#return w

return 0,w

#"The word doesn't exist. Please double check it."

#else:

#return "Sorry, We didn't understand your entry."

# else:

#return "The word doesn't exist. Please double check it."

length = len(split\_sentence)

dest\_lan = 'hi'

#take word from user

#word = input('Enter word: ')

for j in range(length):

word = split\_sentence[j]

meaning, x = getMeaning(word)

if meaning==0:

print("Word doesn't exist")

else:

n =len(meaning)

if n==1:

if x==split\_sentence[j]:

translated\_word = translator.translate(x, src='en', dest = dest\_lan)

print(split\_sentence[j],"(",translated\_word.text," - ",translated\_word.pronunciation,")")

print("Meaning:", meaning[0])

translated\_text = translator.translate(meaning[0], src='en', dest = dest\_lan)

#print(f"The Actual Text was {meaning[0]}")

print(f"The Translated Text is: {translated\_text.text}")

print(f"The Translated Text pronunciation is: {translated\_text.pronunciation}")

print(" ")

else:

translated\_word = translator.translate(x, src='en', dest = dest\_lan)

print(x,"(",translated\_word.text," - ",translated\_word.pronunciation,")")

print("Meaning:", meaning[0])

translated\_text = translator.translate(meaning[0], src='en', dest = dest\_lan)

#print(f"The Actual Text was {meaning[0]}")

print(f"The Translated Text is: {translated\_text.text}")

print(f"The Translated Text pronunciation is: {translated\_text.pronunciation}")

print(" ")

else:

if x==split\_sentence[j]:

translated\_word = translator.translate(x, src='en', dest = dest\_lan)

print(split\_sentence[j], "(",translated\_word.text," - ",translated\_word.pronunciation,")")

print("Meaning:")

for i in range(n):

print("%d.%s"%(i+1,meaning[i]))

translated\_text = translator.translate(meaning[i], src='en', dest = dest\_lan)

#print(f"The Actual Text was {meaning[i]}")

print(f"The Translated Text is: {translated\_text.text}")

print(f"The Translated Text pronunciation is: {translated\_text.pronunciation}")

print(" ")

else:

translated\_word = translator.translate(x, src='en', dest = dest\_lan)

print(x, "(",translated\_word.text," - ",translated\_word.pronunciation,")")

print("Meaning:")

for i in range(n):

print("%d.%s"%(i+1,meaning[i]))

translated\_text = translator.translate(meaning[i], src='en', dest = dest\_lan)

#print(f"The Actual Text was {meaning[i]}")

print(f"The Translated Text is: {translated\_text.text}")

print(f"The Translated Text pronunciation is: {translated\_text.pronunciation}")

print(" ")

**PRE-PROCESSING**

**AHE**:  
import cv2

import numpy as np

from matplotlib import pyplot as plt

image = cv2.imread("C:/Users/bandl/Cropped Image.jpg")

# Automatic brightness and contrast optimization with optional histogram clipping

def automatic\_brightness\_and\_contrast(image, clip\_hist\_percent=1):

gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

# Calculate grayscale histogram

hist = cv2.calcHist([gray],[0],None,[256],[0,256])

hist\_size = len(hist)

# Calculate cumulative distribution from the histogram

accumulator = []

accumulator.append(float(hist[0]))

for index in range(1, hist\_size):

accumulator.append(accumulator[index -1] + float(hist[index]))

# Locate points to clip

maximum = 0.9\*accumulator[-1]

clip\_hist\_percent \*= (maximum/100.0)

clip\_hist\_percent /= 2.0

# Locate left cut

minimum\_gray = 0

while accumulator[minimum\_gray] < clip\_hist\_percent:

minimum\_gray += 1

# Locate right cut

maximum\_gray = hist\_size -1

while accumulator[maximum\_gray] >= (maximum - clip\_hist\_percent):

maximum\_gray -= 1

# Calculate alpha and beta values

alpha = 255 / (maximum\_gray - minimum\_gray)

beta = -minimum\_gray \* 0.9\*alpha

#Calculate new histogram with desired range and show histogram

new\_hist = cv2.calcHist([gray],[0],None,[256],[minimum\_gray,maximum\_gray])

plt.plot(hist)

plt.plot(new\_hist)

plt.xlim([0,256])

plt.show()

auto\_result = cv2.convertScaleAbs(image, alpha=alpha, beta=beta)

return (auto\_result, alpha, beta)

image = cv2.imread("C:/Users/bandl/Cropped Image.jpg")

auto\_result, alpha, beta = automatic\_brightness\_and\_contrast(image)

print('alpha', alpha)

print('beta', beta)

cv2.imshow('auto\_result', auto\_result)

plt.figure()

plt.show()

plt.imshow(image)

plt.figure()

plt.show()

plt.imshow(auto\_result)

plt.title('AHE Image')

plt.show()

cv2.imwrite("AHE Image.jpg", auto\_result)

cv2.waitKey()

**DE-SKEW:**  
import numpy as np

from skimage import io

from skimage.transform import rotate

from skimage.color import rgb2gray

from deskew import determine\_skew

import cv2

import pytesseract

from matplotlib import pyplot as plt

try:

from PIL import Image

except ImportError:

import Image

image1 = Image.open("C:/Users/bandl/Desktop/Pic Repository/blur\_skew1.jpeg")

pytesseract.image\_to\_string(image1, lang ='eng')

def deskew(\_img):

image = io.imread(\_img)

grayscale = rgb2gray(image)

angle = determine\_skew(grayscale)

rotated = rotate(image, angle, resize=True) \* 255

return rotated.astype(np.uint8)

def angleskew(\_imggray):

image1 = io.imread(\_imggray)

grayscale = rgb2gray(image1)

angle = determine\_skew(grayscale)

print("De-skewed Angle: ", angle)

def display\_avant\_apres(\_original):

plt.subplot(1, 2, 1)

plt.imshow(io.imread(\_original))

deskew1= deskew(\_original)

plt.title('Original Image')

plt.subplot(1, 2, 2)

plt.imshow(deskew(\_original))

plt.title('De-skewed Image')

plt.show()

cv2.imwrite("Deskewed Img.jpg", deskew1 )

display\_avant\_apres("C:/Users/bandl/Desktop/Pic Repository/blur\_skew1.jpeg")

angleskew("C:/Users/bandl/Desktop/Pic Repository/blur\_skew1.jpeg")