PROYECTO DE MATERIA TRATAMIENTO DE DATOS

Jaime Andrés Subía

PRE-PROCESAMIENTO DE IMAGENES CON PYTHON

¿PARA QUÉ SIRVE EL PRE-PROCESAMIENTO DE IMÁGENES?

El procesamiento de imágenes se subdivide en **procesamiento análogo y procesamiento digital.** En el procesamiento análogo, las imágenes son manipuladas mediante señales eléctricas, y un ejemplo de ello son las imágenes proyectadas en un televisor, imágenes médicas, entre otros. En el procesamiento digital, se utilizan algoritmos para procesar las imágenes digitales.

De esta forma, el objetivo del procesamiento digital es mejorar las características de las imágenes a través de la eliminación de distorsiones indeseadas, o realzando alguna de sus características, tal que modelos computacionales (eg. IA) pueden optimizarse.

La imagen digital es un arreglo bidimensional de números (o pixeles) entre 0 y 255. Se define como una función matemática f(x,y) en el plano x,y; tal que el valor de la función en cada punto entrega un valor al pixel de la imagen.

La matemática detrás del procesamiento de imágenes es bastante compleja. Afortunadamente, existen paquetes computacionales, como se verá a continuación, que simplifican el proceso en gran medida.

PASOS PRINCIPALES PARA PREPROCESAR UN CONJUNTO DE IMÁGENES

1. Lectura de la imagen
2. Ajuste del tamaño de la imagen
3. Reducción de ruido
4. Segmentación (de ser necesario)
5. Proceso de suavizar la imagen

1. LECTURA DE IMÁGENES

Se almacena la ruta del conjunto de imágenes en una variable, con el objetivo de crear una función que cargue las carpetas conteniendo las imágenes dentro de arreglos

```
#LIBRERIAS INDISPENSABLES

import os
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import cv2
# GLOBAL PATH
image_path = "Path to your dataset"
#FUNCION PARA CREAR ARREGLO
def loadImages(path):
# ORGANIZACION DE ARCHIVOS EN LISTA
image_files = sorted([os.path.join(path, 'train', file)
for file in os.listdir(path + "/train") if file.endswith('.png')])
return image_files
```

2. AJUSTE DE TAMAÑO

Se requiere homogeneizar el tamaño de las imágenes para facilitar la lectura de posteriores algoritmos.

```
# MUESTRA DE IMAGEN ORIGINAL
def display_one(a, title1 = "Original"):
    plt.imshow(a), plt.title(title1)
    plt.xticks([]), plt.yticks([])
    plt.show()# Display two images
def display(a, b, title1 = "Original", title2 = "Edited"):
    plt.subplot(121), plt.imshow(a), plt.title(title1)
    plt.xticks([]), plt.yticks([])
    plt.subplot(122), plt.imshow(b), plt.title(title2)
    plt.xticks([]), plt.yticks([])
    plt.show()# Preprocessing
def processing(data):
    # CARGANDO 3 IMAGENES
    img = [cv2.imread(i, cv2.IMREAD_UNCHANGED) for i in data[:3]]
print('Original size',img[0].shape)
    # RESIZE DIMENSIONES
    height = 220
    width = 220
    dim = (width, height)
    res_img = []
    for i in range(len(img)):
        res = cv2.resize(img[i], dim,
interpolation=cv2.INTER_LINEAR)
        res_img.append(res)
    # CHECK
    print("RESIZED", res_img[1].shape)
```

```
    # VISUALIZACION
    original = res_img[1]
    display_one(original)
```

3. REDUCCION DE RUIDO (procesamiento morfológico y filtro gaussiano)

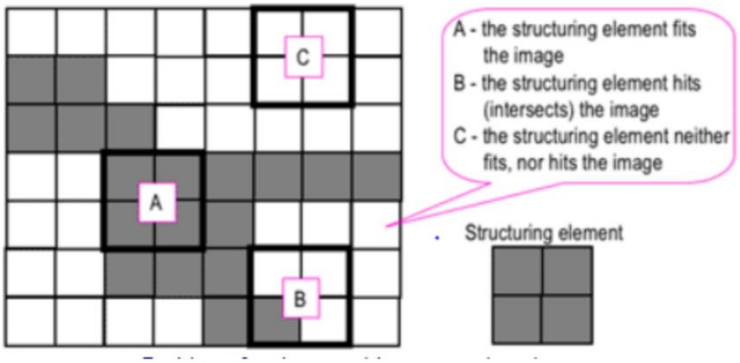
Se puede utilizar, por ejemplo, un filtro de tipo gaussiano para incrementar el *blur*, o que tan borrosa se ve la imagen, mediante una función de tipo gaussiano. Puede utilizarse para realizar estructuras dentro de la imagen bajo distintas escalas.

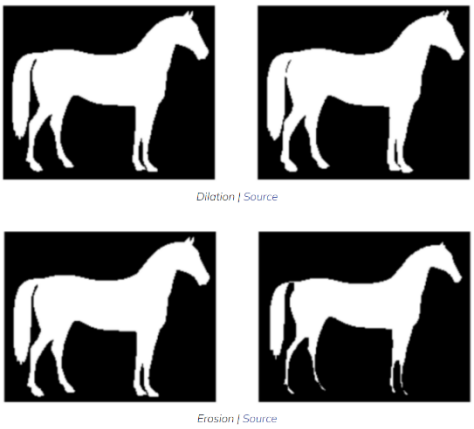
```
#                       Gaussian
no_noise          =          []
for  i  in  range(len(res_img)):
blur                          =
cv2.GaussianBlur(res_img[i],
(5,            5),            0)
no_noise.append(blur)
image       =       no_noise[1]
display(original,         image,
'Original', 'Blured')
```

2. Procesamiento de imágenes morfológico: trata de eliminar las imperfecciones de las imágenes binarias y ayuda a suavizar la imagen mediante operaciones de apertura y cierre.

Las operaciones morfológicas se pueden extender a imágenes en escala de grises. Consiste en operaciones no lineales relacionadas con la estructura de características de una imagen. Depende del orden relacionado de los píxeles pero de sus valores numéricos. Esta técnica analiza una imagen utilizando una pequeña plantilla conocida como elemento estructurador que se coloca en diferentes ubicaciones posibles en la imagen y se compara con los píxeles de vecindad correspondientes. Un elemento estructurante es una pequeña matriz con valores 0 y 1.

Las dos operaciones fundamentales del procesamiento de imágenes morfológicas, Dilatación y Erosión: la operación de dilatación agrega píxeles a los límites del objeto en una imagen. La operación de erosión elimina los píxeles de los límites del objeto.

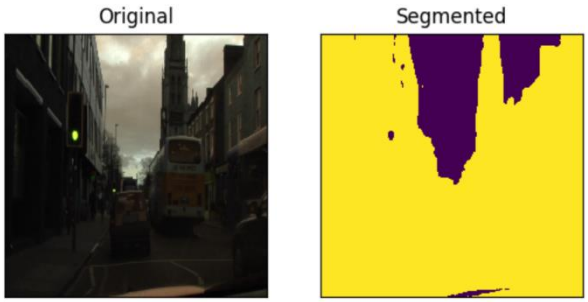




4. SEGMENTACION Y MORFOLOGÍA

Consiste en la separación del fondo de la imagen y los objetos en ella.

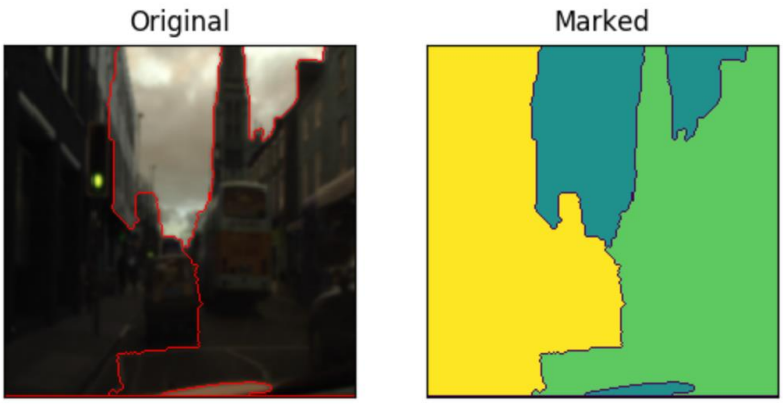
```
# Segmentation
gray = cv2.cvtColor(image,
cv2.COLOR_RGB2GRAY)
ret, thresh =
cv2.threshold(gray, 0, 255,
cv2.THRESH_BINARY_INV +
cv2.THRESH_OTSU)

# Displaying segmented
images
display(original, thresh,
'Original', 'Segmented')
```



Posteriormente, se puede proceder a la aplicación de otro *blurring,* y posteriormente se separa a los objetos mediante marcadores:

```
# Marker labelling
ret, markers = cv2.connectedComponents(sure_fg)

# Add one to all labels so that sure background is not 0, but 1
markers = markers + 1

# Now, mark the region of unknown with zero
markers[unknown == 255] = 0

markers = cv2.watershed(image, markers)
```

```
image[markers == -1] = [255, 0, 0]

# Displaying markers on the image
display(image, markers, 'Original', 'Marked')
```



OTROS PAQUETES

OpenCV es una biblioteca (paquete de código abierto preconstruida que se usa ampliamente para aplicaciones de procesamiento de imágenes, ML y visión por computadora. Admite una buena variedad de lenguajes de programación, incluido Python. Permite leer, reestablecer tamaño, extraer planos por pixeles y reconstruir imágenes, y construir imágenes sintéticas.

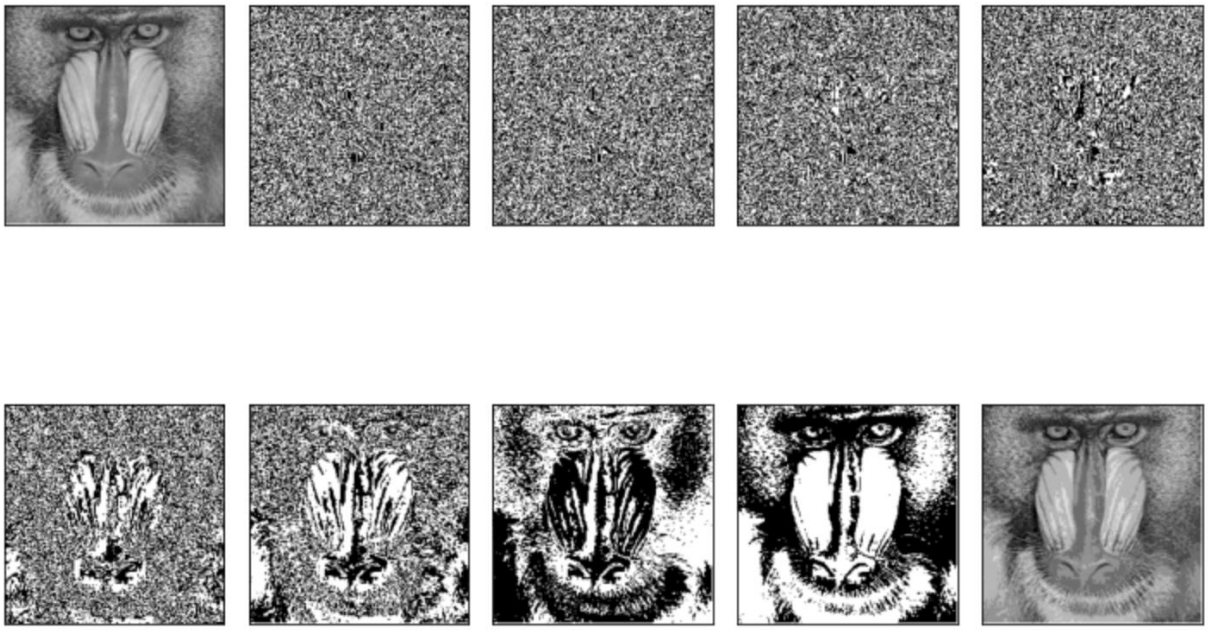
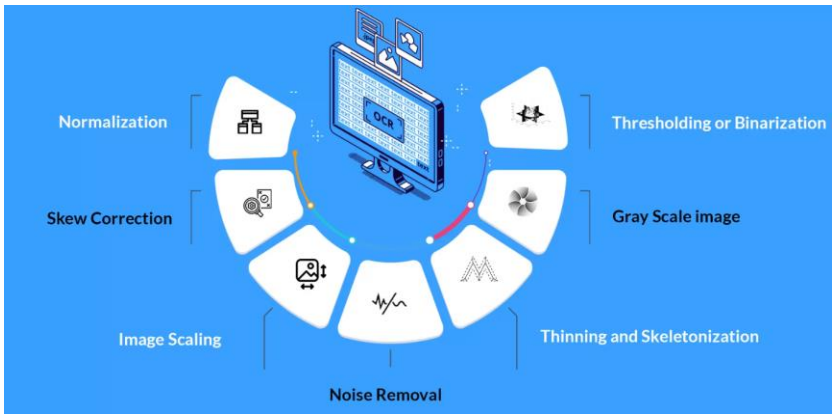


Figura. Reconstrucción de imagen por medio de OpenCV. Código completo en: https://www.analyticsvidhya.com/blog/2021/09/a-beginners-guide-to-image-processing-with-opencv-and-python/

APLICACIÓN: OCR

El reconocimiento de caracteres ópticos (OCR por sus siglas en inglés) es una de las aplicaciones más importantes del preprocesamiento de imágenes. En OCR, se extrae el texto dentro de imágenes por lo cual puede utilizarse para el manejo de información y documentos tanto físicos como electrónicos. Las imágenes que alimentan el algoritmo de OCR deben estar apropiadamente pre procesadas mediante procesos como el que se visualiza a continuación:



Lo cual puede llevarse a cabo mediante paquetes como OpenCV, introducido previamente.

Un ejemplo de ello se visualiza en el siguiente recurso online: https://www.nextgeninvent.com/7-steps-of-image-pre-processing-to-improve-ocr-using-python/?utm_source=rss&utm_medium=rss&utm_campaign=7-steps-of-image-pre-processing-to-improve-ocr-using-python

REFERENCIAS

Practical Machine Learning and Image Processing: For Facial Recognition, Object Detection, and Pattern Recognition Using Python. Singh, H. 2019. Apress

Image preprocessing. Medium. Available in: https://prince-canuma.medium.com/image-pre-processing-c1aec0be3edf. Google collab project full length: https://colab.research.google.com/github/Blaizzy/BiSeNet-Implementation/blob/master/Preprocessing.ipynb

Image Processing in Python: Algorithms, Tools, and Methods You Should Know. Neetika Khandelwal. https://neptune.ai/blog/image-processing-python

OPENCV official website. Available in: https://docs.opencv.org/4.5.2/index.html