TAREA 5: USO DE LIBRERIA OPENCV PARA PROCESAMIENTO DE IMAGENES

Marcos Rafael Roman Salgado[1]

[1]*UACh, Texcoco de Mora – <marcosrafael2000@hotmail.com>*

RESUMEN

Esta práctica involucra la aplicación de varios filtros de procesamiento de imágenes utilizando OpenCV en Google Colab. Las operaciones incluyen convertir una imagen a escala de grises, suma y resta de imágenes, cambio de tamaño, ecualización de histograma, filtros convolucionales (promedio, gaussiano, realce, Sobel), umbralización (Otsu, simple, adaptativa) y operadores morfológicos (erosión, dilatación, apertura, cierre, momentos de Hu). El objetivo es comprender estas técnicas de preprocesamiento como un preludio al estudio de redes neuronales convolucionales (CNN).

Palabras-clave: *Procesamiento de Imágenes, OpenCV, Google Colab, Aplicaciones Agrícolas, Visión por Computadora*

ABSTRACT

This practice involves the application of various image processing filters using OpenCV in Google Colab. The operations include converting an image to grayscale, image addition and subtraction, resizing, histogram equalization, convolutional filters (mean, Gaussian, sharpening, Sobel), thresholding (Otsu, simple, adaptive), and morphological operators (erosion, dilation, opening, closing, Hu moments). The objective is to understand these preprocessing techniques as a precursor to studying convolutional neural networks (CNNs).

Keywords: *Image Processing, OpenCV, Google Colab, Agricultural Applications, Computer Vision*

1. INTRODUCCIÓN

El procesamiento de imágenes es una etapa crucial en la preparación de datos para redes neuronales convolucionales (CNNs). En esta práctica, utilizamos diversas técnicas de procesamiento de imágenes implementadas en la biblioteca OpenCV para familiarizarnos con los conceptos y métodos clave. OpenCV es una poderosa herramienta que proporciona numerosas funciones para la manipulación y análisis de imágenes. Esta práctica sirve como una preparación para el siguiente tema en nuestro diplomado, que es el estudio de las redes neuronales convolucionales.

2. MATERIALES Y MÉTODOS

La práctica se llevó a cabo utilizando Google Colab, una plataforma en línea que permite ejecutar código Python en notebooks. La biblioteca OpenCV se utilizó para aplicar varios filtros y operaciones de procesamiento de imágenes. A continuación se describen las operaciones realizadas:

2.1. Conversión a Escala de Grises

Se utilizó la función `cv2.cvtColor` para convertir una imagen de color a escala de grises.

2.2. Suma y Resta de Imágenes

Se aplicaron las funciones `cv2.add` y `cv2.subtract` para sumar y restar dos imágenes, respectively.

2.3. Cambio de Tamaño de Imagen

La función `cv2.resize` se utilizó para cambiar el tamaño de las imágenes.

2.4. Ecualización del Histograma

Se implementó la ecualización del histograma usando `cv2.equalizeHist` para mejorar el contraste de las imágenes en escala de grises.

2.5. Filtros Convolucionales

Se aplicaron varios filtros convolucionales:

- Filtro Promedio: `cv2.blur`
- Filtro Gaussiano: `cv2.GaussianBlur`
- Filtro de Realce: Utilizando un kernel de realce
- Filtro Sobel: `cv2.Sobel`

2.6. Umbralización (Thresholding)

Se aplicaron diferentes técnicas de umbralización:

- Umbralización Simple: `cv2.threshold`
- Umbralización de Otsu: `cv2.threshold` con `cv2.THRESH_OTSU`
- Umbralización Adaptativa: `cv2.adaptiveThreshold`

2.7. Operadores Morfológicos

Se utilizaron los operadores morfológicos básicos:

- Erosión: `cv2.erode`
- Dilatación: `cv2.dilate`
- Apertura: `cv2.morphologyEx` con `cv2.MORPH_OPEN`
- Cierre: `cv2.morphologyEx` con `cv2.MORPH_CLOSE`
- Momentos de Hu: `cv2.HuMoments`

3. RESULTADOS

Los resultados obtenidos tras la aplicación de los diferentes filtros y técnicas de procesamiento de imágenes se describen a continuación:

3.1. Conversión a Escala de Grises

La imagen convertida a escala de grises mostró una representación en tonos de gris de la imagen original, facilitando su análisis posterior.

3.2. Suma y Resta de Imágenes

La suma y resta de imágenes resultó en combinaciones de las imágenes originales, destacando ciertas características o eliminando otras.

3.3. Cambio de Tamaño de Imagen

Las imágenes redimensionadas se adaptaron a las dimensiones especificadas, manteniendo las proporciones o ajustándose según fue necesario.

3.4. Ecualización del Histograma

La ecualización del histograma mejoró significativamente el contraste de las imágenes, destacando detalles previamente ocultos.

3.5. Filtros Convolucionales

- Filtro Promedio: Suavizó la imagen reduciendo el ruido.

Figura 1. Imagen a escala de grises.

Fuente: Impartida por profesor.

Figura 2. Suma de imagenes.

Fuente: Impartida por profesor.

- Filtro Gaussiano: Redujo el ruido manteniendo las estructuras importantes.
- Filtro de Realce: Aumentó la nitidez de la imagen.
- Filtro Sobel: Destacó los bordes de la imagen.

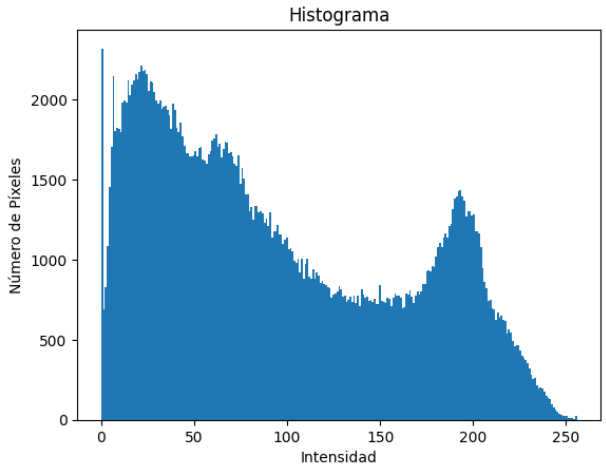
3.6. Umbralización (Thresholding)

- Umbralización Simple: Binarizó la imagen con un umbral fijo.

Figura 3. Imagen escalada.

Fuente: Impartida por profesor.

Figura 4. Histograma resultante de una imagen.



Fuente: Impartida por profesor.

- Umbralización de Otsu: Binarizó la imagen ajustando el umbral automáticamente.
- Umbralización Adaptativa: Binarizó la imagen considerando variaciones locales.

3.7. Operadores Morfológicos

- Erosión: Redujo el tamaño de los objetos blancos en la imagen.
- Dilatación: Aumentó el tamaño de los objetos blancos en la imagen.
- Apertura: Eliminó pequeños objetos blancos del fondo.
- Cierre: Rellenó pequeños agujeros en los objetos blancos.
- Momentos de Hu: Proporcionaron características invariantes para el reconocimiento de formas.

4. DISCUSIÓN

La aplicación de diferentes técnicas de procesamiento de imágenes con OpenCV proporciona una comprensión sólida de cómo se pueden mejorar y analizar las imágenes antes de introducirlas en una red neuronal convolucional (CNN). Cada filtro y operación tiene un impacto específico en la imagen, resaltando diferentes características y mejorando la calidad general para su análisis posterior. La comprensión de estos filtros es crucial para el preprocesamiento de datos en tareas de visión por computadora, especialmente en aplicaciones agrícolas donde la calidad de la imagen puede variar considerablemente.

5. CONCLUSIONES

En esta práctica, se ha demostrado cómo utilizar varias técnicas de procesamiento de imágenes con OpenCV en Google Colab. Las operaciones realizadas permiten una mejor comprensión y manipulación de las imágenes, preparando los datos para su uso en redes neuronales convolucionales (CNNs). La capacidad de preprocesar imágenes de manera efectiva es fundamental en la visión por computado-

Figura 5. Imagen con convolución.

Fuente: Impartida por profesor.

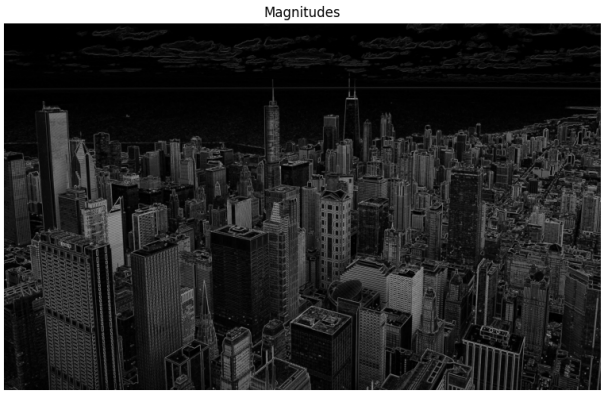
Figura 6. Imagen con filtro gaussiano.

Fuente: Impartida por profesor.

Figura 7. Imagen con realce

Fuente: Impartida por profesor.

Figura 8. Imagen con filtro sobel

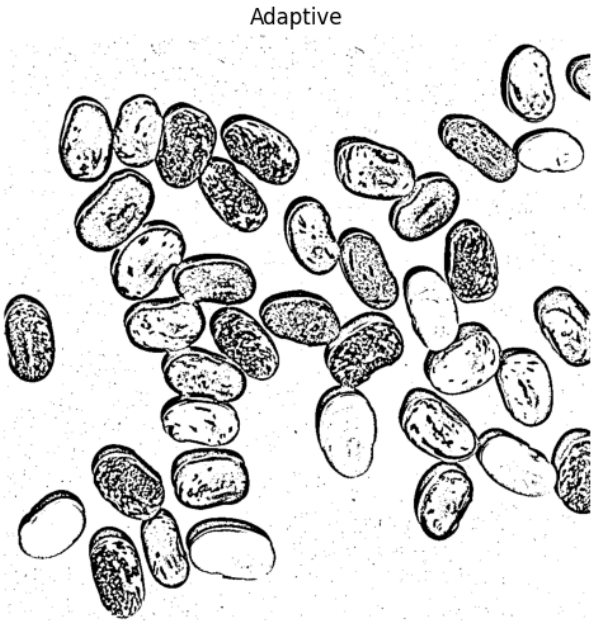


Fuente: Impartida por profesor.

Figura 9. Imagen con umbralizacion de otsu.

Fuente: Impartida por profesor.

Figura 10. Imagen con umbralizacion adaptativa.



Fuente: Impartida por profesor.

ra y tiene aplicaciones directas en la agricultura de precisión.

APÉNDICES

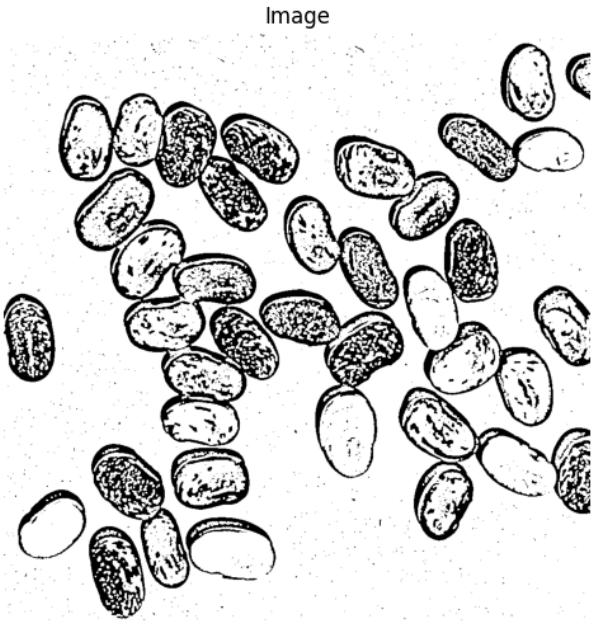
Código en Google Colab

A continuación se muestra el código Python utilizado en Google Colab para realizar las operaciones

Figura 11. Imagen con Erosion.

Fuente: Impartida por profesor.

Figura 12. Imagen con apertura.



Fuente: Impartida por profesor.

de procesamiento.

Código 1. Codigo de Python.

```python
# Importamos la biblioteca OpenCV.
import cv2
import matplotlib.pyplot as plt
import numpy as np

# Leer la imagen desde un archivo. Reemplaza 'ruta_de_tu_imagen.jpg' con la ruta de tu imagen.
imagen = cv2.imread('/content/Manzana.jpg')

# Verificar si la imagen se cargó correctamente.
if imagen is None:
    print("Error: No se pudo cargar la imagen.")
    exit()

# Convertir la imagen a escala de grises.
imagen_gris = cv2.cvtColor(imagen, cv2.COLOR_BGR2GRAY)

imagen_1 = cv2.imread("Manzana.jpg",0)
imagen_2 = cv2.imread("Manzana.jpg",0)

# Asegurarse de que ambas imágenes tengan las mismas dimensiones
if imagen_1.shape != imagen_2.shape:
    print("Las imágenes deben tener las mismas dimensiones")
    exit()

SumA = cv2.add(imagen_1, imagen_2)

img = cv2.imread('Manzana.jpg')

print(f'Imagen original: {img.shape}')

# Define el porcentaje de escala al que se desea redimensionar la imagen.
# En este caso, la imagen se reducirá al 20% de su tama o original.
scale_percent = 20

# Calcula las nuevas dimensiones de la imagen:
# - width: el nuevo ancho, que es el 20% del ancho original.
# - height: la nueva altura, que es el 20% de la altura original.
```

```
width = int(img.shape[0] * scale_percent / 100)
height = int(img.shape[1] * scale_percent / 100)

# Agrupa las dimensiones en una tupla.
dim = (width, height)

# Redimensiona la imagen original a las dimensiones especificadas usando interpolación lineal.
# El resultado se almacena en 'img_SCALED'.
img_SCALED = cv2.resize(img, dim, interpolation= cv2.INTER_CUBIC)

# HIstograma
plt.hist(img.ravel(), bins = 255 , range=[0,256])
plt.title(title)
plt.xlabel('Intensidad')
plt.ylabel('Número de Píxeles')
plt.show()

# Definimos un kernel (filtro) 3x3. Este filtro es básicamente un promedio 3x3.
kernel = np.array((
  [1, 1, 1],
  [1, 1, 1],
  [1, 1, 1]), dtype="float32") / 9

# Aplicamos el filtro a la imagen usando la función filter2D de OpenCV.
# El -1 indica que la imagen de salida tendrá la misma profundidad que la imagen de entrada.
outputFiltro = cv2.filter2D(img, -1, kernel)

# Aplicar filtro gaussiano
# El segundo argumento es el tama o del kernel. Debe ser impar y puede ser diferente en x e y.
# El tercer argumento es la desviación estándar en la dirección X.
# Si se establece en 0, se calcula a partir del tama o del kernel.
img_gaussiana = cv2.GaussianBlur(img, (5,5), 0)


# Kernel para el realce
kernel_sharpen = np.array((
    [-1, -1, -1],
    [-1,  9, -1],
    [-1, -1, -1]), dtype="float32")

# Aplicar el kernel usando filter2D
img_enhanced = cv2.filter2D(img, -1, kernel_sharpen)

img = cv2.imread('ciudad.jpg', 0)
img = cv2.GaussianBlur(img, (5,5), 0)
# Kernel Sobel para detectar bordes horizontales
sobel_horizontal = cv2.Sobel(img, cv2.CV_64F, 0, 1, ksize= 3)

# Kernel Sobel para detectar bordes verticales
sobel_vertical = cv2.Sobel(img, cv2.CV_64F, 1, 0, ksize=3)

# Calcular la magnitud de los gradientes
magnitude = cv2.magnitude(sobel_horizontal, sobel_vertical)

# @title  { run: "auto" }
# Aplicar un umbral a la imagen
umbral = 29 # @param {type:"slider", min:0, max:255, step:1}

# Supongamos que 'blurred' es tu imagen preprocesada con un filtro de desenfoque
# Por ejemplo, si usaste un filtro Gaussiano:
# blurred = cv2.GaussianBlur(image, (5, 5), 0)

#Umbralizacion

(T, thresh) = cv2.threshold(blurred, umbral, 255, cv2.THRESH_BINARY)

(T, otsuThresh) = cv2.threshold(
```

```
    blurred, 0, 255, cv2.
    THRESH_BINARY | cv2.THRESH_OTSU)
106
107
108 meanAdaptiveThresh = cv2.
    adaptiveThreshold(img, 255, cv2.
    ADAPTIVE_THRESH_MEAN_C, cv2.
    THRESH_BINARY_INV, 21, 10)
109
110 gaussianAdaptiveThresh = cv2.
    adaptiveThreshold(img, 255, cv2.
    ADAPTIVE_THRESH_GAUSSIAN_C, cv2.
    THRESH_BINARY, 21, 5)
111
112
113 #Operadores morfologicos
114
115 # Crear un kernel de 3x3 de tipo
    uint8
116 kernel = np.ones((3,3), np.uint8)
117
118 # Aplicar la operación de erosión a
     la imagen '
    gaussianAdaptiveThresh'
119 eroded_image = cv2.erode(
    gaussianAdaptiveThresh, kernel,
    iterations=1)
120
121 # Crear un kernel de 3x3 de tipo
    uint8
122 kernel = np.ones((3,3), np.uint8)
123
124 # Aplicar la operación de dilatació
    n a la imagen '
    gaussianAdaptiveThresh'
125 dilated_image = cv2.dilate(
    gaussianAdaptiveThresh, kernel,
    iterations=1)
126
127 kernel = np.ones((3,3), np.uint8)
128 opened_image = cv2.morphologyEx(
    gaussianAdaptiveThresh, cv2.
    MORPH_OPEN, kernel)
129
130 kernel = np.ones((3,3), np.uint8)
131 closed_image = cv2.morphologyEx(
    gaussianAdaptiveThresh, cv2.
    MORPH_CLOSE, kernel)
132
133 # Momentos Hu
134 # Suponiendo que tienes una imagen
    binarizada 'binary_image'
135 contours, _ = cv2.findContours(
    otsuThresh, cv2.RETR_EXTERNAL,
    cv2.CHAIN_APPROX_SIMPLE)
136
137 hu_moments_list = []
138
139 for contour in contours:
140     # Calcula los momentos normales
    del contorno
141     moments = cv2.moments(contour)
142
143     # Calcula los momentos de Hu a
    partir de los momentos normales
144     hu_moments = cv2.HuMoments(
    moments)
145
146     # Opcional: si quieres
    transformar los momentos para
    mejorar la comparación
147     log_hu_moments = [-np.sign(m)*
    np.log10(np.abs(m)) for m in
    hu_moments]
148
149     hu_moments_list.append(
    log_hu_moments)
```

Contacto

Para cualquier duda favor de contactar:

marcosrafael2000@hotmail.com

+52 5537149673

Referencias

OpenCV-Python Tutorials Documentation. (2024). *OpenCV-Python Tutorials*. Recuperado de <https://docs.opencv.org/4.x/d6/d00/tutorial_py_root.html>

Laganière, R. (2011). *OpenCV 2 Computer Vision Application Programming Cookbook*. Packt Publishing Ltd.

Gonzalez, R. C., Woods, R. E. (2018). *Digital Image Processing*. Pearson.