Clasificación morfológica foliar mediante procesamiento de imágenes

Jesus Rodolfo Izurieta Veliz

25 de julio de 2020

Resumen

El reconocimiento de objetos en imágenes y su clasificación es un problema ampliamente estudiado en la actualidad, y cuando es necesario aplicarlo, en general suele pensarse en una solución que implemente redes neuronales y una gran cantidad de imágenes de prueba para entrenarlas, sin embargo, en algunos casos específicos, este problema puede tener soluciones más sencillas y rápidas de implementar. En este caso estudiamos el problema de la clasificación morfológica de hojas de plantas, sin recurrir al uso de redes neuronales y con un número reducido de imágenes de prueba.

1. Introducción

Las redes neuronales son en general una solución eficiente para el problema de reconocimiento de objetos en imágenes, especialmente en casos en que el objeto que se quiere obtener de una imagen puede encontrarse en muchas posiciones y tener una gran cantidad de variaciones, para lo cual es necesaria una gran cantidad de imágenes de prueba de modo que la red neuronal pueda entrenarse y responder apropiadamente a una gran cantidad de situaciones. En este caso, el problema que tratamos de resolver es de una complejidad menor y puede resolverse por otros medios.

Para clasificar la hoja de una planta, necesitamos una forma de representar su forma, en este punto nos será útil el método de momentos invariantes de Hu, que genera valores que describen la forma de un objeto si tenemos su silueta. Los momentos invariantes de Hu funcionan correctamente en objetos en dos dimensiones y pueden darnos valores de una imagen que no varían ante transformaciones simples como escalado o rotación, sin embargo, no nos serán útiles para transformaciones en tres dimensiones, por lo tanto, su aplicación en este problema lo convierte

en una buena opción, ya que en general, para observar correctamente la hoja de una planta, suele acomodársela de modo que su forma se puede observar claramente.

Como un proceso previo a la clasificación de una hoja por la forma que tiene, tendremos que encontrar la manera de obtener su silueta de la forma más limpia posible, por lo que utilizamos también técnicas de preprocesado de imágenes como binarización de imágenes Otsu, que nos permite separar objetos en una imagen según la disposición del histograma de la imagen.

1.1. Objetivos

1.1.1. Objetivo general

- Desarrollo de una biblioteca para reconocer y clasificar hojas de plantas en imágenes sin hacer uso de métodos avanzados de inteligencia artificial como redes neuronales.

1.1.2. Objetivos específicos

- Desarrollo de una biblioteca que permita una cómoda manipulación de imágenes para su procesamiento.
- Expansión de la biblioteca de procesamiento de imágenes implementando diferentes algoritmos de preprocesamiento de imágenes.
- Desarrollo de un notebook destallando paso a paso el uso de la biblioteca para este caso específico.

2. Metodología

Este trabajo constituye un análisis descriptivo de la aplicación de diversos métodos de procesamiento de imágenes, que son aplicados a este problema específico para comprobar sus resultados. Se parte de la hipótesis inductiva de que las técnicas usadas nos servirán para conseguir los objetivos de este proyecto. Estas técnicas que fueron diseñadas para diferentes etapas del procesamiento de imágenes, pueden aplicarse secuencialmente, ya que cada una corresponde a una tarea específica que será complementada por las demás.

Seguimos entonces una secuencia de acciones con la finalidad de cumplir el objetivo de clasificar una hoja en una imagen digital, siguiendo los pasos como se muestra a continuación:

Binarización de la imagen $\rightarrow$ **Descripción morfológica** $\rightarrow$ **Clasificación** $\rightarrow$ **Comparación**

Cada uno de estos pasos implementa de forma independiente una técnica de procesamiento de imágenes, por lo tanto, la imagen de prueba pasará de forma secuencial por cada uno de estos procesos, siendo la salida de uno, la entrada del siguiente.

El proceso de binarización de una imagen nos permite separar los objetos de interés de una fotografía, para esto usamos la técnica de filtrado Otsu, que analiza el histograma de la imagen encontrando un punto medio desde el cual podremos separar grupos de objetos por colores. Para la descripción morfológica, usamos momentos invariantes de Hu, una técnica que permite describir la forma de una imagen mediante valores numéricos que no varían ante transformaciones simples como escalado o rotación. Se analizan estadísticamente dos momentos de Hu ($\phi 1$ y $\phi 2$) para cada tipo de hoja de un conjunto de imágenes de prueba, con la finalidad de poder determinar áreas en el plano $\phi 1$, $\phi 2$ en que se encuentre cada tipo de hoja. Finalmente comparamos la posición en el plano de los momentos obtenidos de una imagen para poder clasificarla en uno de estos grupos con cierta probabilidad de pertenencia según su distancia al centro de gravedad de cada conjunto.

3. Morfología foliar

La morfología foliar ...

4. Desarrollo

El proyecto no contará con una interfaz gráfica, sino que constará de un notebook interactivo de jupyter en el que se podrá apreciar tanto el código fuente como los resultados de cada paso del proceso de manipulado de las imágenes. Para esto, se desarrollan las siguientes bibliotecas:

4.1. Representación de imágenes

Primeramente definiremos algunos conceptos que utilizaremos en el desarrollo del proyecto. Para el tratamiento de una imagen digital, la representaremos como una función de la posición de sus pixeles como sigue:

$$I : \mathbb{N}^2 \to [0, 1]$$

$$I(x, y)$$

Donde el rango representa el nivel de luminosidad para los pixeles en coordenadas x e y en el caso de imágenes en escala de grises, para imágenes a color, tendremos en su lugar una tupla de tres elementos, uno para cada uno de los colores rojo, verde y azul. En el caso de una imagen digital, estos valores varían en el rango $[0, 255] \in \mathbb{N}$.

4.2. Biblioteca Image

La clase imagen es una abstracción básica que nos permitirá acceder y manipular fácilmente una imagen que puede ser cargada desde un archivo, o puede ser creada desde un array y posteriormente mostrada en un notebook de jupyter.

Load file: (Parámetros: ruta) Carga en la instancia la imagen que se encuentra en la ruta y la convierte en una matriz de tuplas de 3 componentes, uno por cada color RGB, almacenándola en el atributo `array` de la clase imagen, además define otros atributos como las dimensiones de la imagen.

Load array: (Parámetros: array) Almacena una matriz en el atributo `array` de la imagen, la matriz debe ser una matriz de tuplas de tres valores.

Show: Retorna un objeto PIL.Image.Image que permite visualizar la imagen en un notebook de jupyter o almacenarla como un archivo.

I: (Parámetros: x, y) Representa la función $I(x, y)$ definida previamente, retorna los píxeles en la posición x, y como una tupla de 3 valores.

I normal: (Parámetros: x, y) Igual que I() pero en lugar de retornar valores en el rango [0, 255] normaliza los valores al rango [0, 1].

I m: (Parámetros: x, y, color) Igual que I(), pero retorna un único valor definido por el parámetro `color`, 0 para rojo, 1 para verde y 2 para azul, el valor por defecto es rojo si no se define el parámetro color.

I mnormal: (Parámetros: x, y, color) Igual que $I_m()$ pero con valores normalizados en el rango [0, 1]

Iterator: Retorna un iterador de tuplas x, y que facilita iteraciones sobre cada píxel de la imagen.

Map over: (Parámetros: func) Permite sobreescribir píxeles mediante una función que se envía como parámetro, la función recibirá una tupla de 3 valores y debe devolver una tupla de 3 valores. Esta función nos permitirá recorrer la totalidad de la imagen aplicando en cada

pixel la función `func`, por ejemplo el código `img.map_over(lambda x, y, z: (x, x, x))` ejecutado sobre una imagen img2, permitirá cambiar el valor de cada canal por su valor rojo, lo que convertirá a la imagen a blanco y negro.

4.3. Preprocesamiento

4.3.1. Reducción de ruido

4.3.2. Corrección de iluminación

4.3.3. Segmentación

4.4. Procesamiento

4.4.1. Filtrado Otsu

4.4.2. Momentos de Hu

Para obtener un valor propio de la forma geométrica de un objeto en una imagen, usaremos momentos invariantes de Hu, un conjunto de valores calculados en base a la imagen, que nos permiten describir su forma y que permanecen constantes aún cuando se aplican sobre la imagen transformaciones como rotada o escalado.

Los momentos de orden $(p+q)$ en dos dimensiones sobre una función de distribución $p(x,y)$ continua, son calculados mediante integrales de Riemman como sigue:

$$m_{pq} = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} x^p y^q p(x,y)\,dx\,dy \quad p,q = 0,1,2,...$$

En este caso, usaremos momentos centrales que se definen como sigue para una función $p(x,y)$ continua:

$$\mu_{pq} = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} (x-\bar{x})^p (y-\bar{y})^q p(x,y)\,d(x-\bar{x})\,d(y-\bar{y})$$

donde:

$$\bar{x} = \frac{m_{10}}{m_{00}}, \quad \bar{y} = \frac{m_{01}}{m_{00}}$$

son las coordenadas del centroide del objeto en la imagen.

Sin embargo, ya que se trata de una imagen digital en que usamos valores discretos, calcularemos los momentos centrales con las ecuaciones:

$$\mu_{pq} = \sum_x \sum_y x^p y^q I(x,y)$$

$$n_{pq} = \frac{\mu_{pq}}{\mu_{00}^r}, \quad r = \frac{p+q}{2} + 1$$

Con estas ecuaciones podremos calcular los momentos invariantes que usaremos en este proyecto. Nos limitamos a usar los primeros dos momentos invariantes de Hu, ya que nos son suficientes en este caso.

$$\phi_1 = n_{20} + n_{02}, \quad \phi_2 = (n_{20} + n_{02})^2 + 4n_{11}^2$$

Estos dos valores representan la extensión y la estrechez del objeto respectivamente[1].

4.4.3. Datos de comparación

4.4.4. Comparación y clasificación

4.5. Resultados

4.6. Evaluación

5. Conclusiones

6. Referencias

7. Anexos

7.1. Código fuente

7.1.1. Biblioteca Imagen

```
#!/usr/bin/env python3
import matplotlib.image as plt_img
import numpy as np
from PIL import Image as pil_image
from functools import reduce
from math import sqrt
```

[1] *spread* y *slenderness* en el paper original.

```
class Image:

    def __init__(self):
        self.route = "."
        self.image = None
        self.array = None
        self.height, self.width, self.dat = (0,0,0)

    def load_file(self, route):
        self.route = route
        self.image = plt_img.imread(route)
        self.array = self.image.tolist()
        self.height, self.width, self.dat = self.image.shape

    def load_array(self, array):
        self.array = array
        self.height = len(array)
        self.width = len(array[0])

    def I(self, x, y):
        """ Devuelve rgb en x, y """
        return tuple(self.array[x][y])

    def I_m(self, x, y, color=0):
        """ Devuelve un color de x, y """
        triple = self.array[x][y]
        return triple[color]

    def I_normal(self, x, y):
        """ Devuelve rgb en [0, 1] """
        (i, j, k) = self.I(x, y)
        return (i/255, j/255, k/255)
```

```
def I_mnormal(self, x, y, color=0):
    """ Devuelve color en [0, 1] """
    i = self.I_m(x, y, color)
    return i/255

def show(self, route=None):
    if route:
        self.route = route
    image_arr = np.asarray(self.array, dtype="uint8")
    img_file = pil_image.fromarray(image_arr, 'RGB')
    return img_file

def iterator(self):
    for i in range(self.height):
        for j in range(self.width):
            yield (i,j)

def map_over(self, func):
    """ Ejecución de func sobre cada pixel """
    for x, y in self.iterator():
        self.array[x][y] = func(*self.I(x, y))

def crop(self, x1, y1, x2, y2):
    cropped_array = []
    for i in range(y1, y2):
        arr = []
        for j in range(x1, x2):
            arr.append(self.I(i, j))
        cropped_array.append(arr)
    cropped_image = Image()
    cropped_image.load_array(cropped_array)
    return cropped_image
```

```
def hu_moments(self):
    """ Primeros dos momentos de Hu """
    def moment_pq(p, q):
        """ Momentos geométricos """
        sum = 0
        for x, y in self.iterator():
            sum += x**p * y**q * self.I_mnormal(x, y)
        return sum

    m00 = moment_pq(0, 0)
    m01 = moment_pq(0, 1)
    m11 = moment_pq(1, 1)
    m10 = moment_pq(1, 0)
    m20 = moment_pq(2, 0)
    m02 = moment_pq(0, 2)

    def central_moment_20(a, b, c):
        """ Momentos centrales """
        return (a-(b**2/c))/(c**2)

    n20 = central_moment_20(m20, m10, m00)
    n02 = central_moment_20(m02, m01, m00)
    n11 = central_moment_20(m11, sqrt(m10*m01), m00)

    self.X, self.Y = (n20+n02, (n20-n02)**2+4*(n11**2))

    return (self.X, self.Y)
```

7.2. Pruebas y resultados

7.2.1. Momentos invariantes de Hu