

Clasificación automática de tipos de hojas en imágenes

Jesus Rodolfo Izurieta Veliz

23 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 estudiaremos el problema de la clasificación automática de tipos de hojas, 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.

2. Objetivos

2.1. 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.

2.2. 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.

3. Fundamentos teóricos

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 píxeles como sigue:

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

$$I(x, y)$$

Donde el rango representa el nivel de luminosidad para los píxeles 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. 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. Clase 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 `Im()` 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 sobrescribir 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.

4.2. Preprocesamiento

4.2.1. Reducción de ruido

4.2.2. Corrección de iluminación

4.3. Segmentación

4.4. Procesamiento

4.4.1. Momentos de Hu

Para obtener un valor propio de la forma geométrica de un objeto en una imagen, usaremos momentos de Hu o momentos invariantes, un algoritmo que nos permite obtener un conjunto de valores asignados a una matriz según la disposición que esta tenga, sin variar en cuanto a transformaciones como escalado o rotación.

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:

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

Sin embargo, en el caso del tratamiento de imágenes, esta será la función discreta $I(x, y)$ que describa nuestra imagen en función de la posición de sus pixeles, por lo que la definición de momentos usada será:

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

4.4.2. Extracción de rasgos**4.4.3. Datos de comparación****4.4.4. Comparación y clasificación****5. Pruebas****6. Conclusiones****7. Referencias****8. Anexos****8.1. Código fuente****8.1.1. Clase Imagen****8.1.2. Filtrado ADF****8.1.3. Filtrado Top-Hat****8.1.4. Filtrado Otsu****8.1.5. Momentos de Hu**

Obtención de momentos geométricos de orden $p + q$, $m_{pq} = \sum_x \sum_y x^p y^q I(x, y)$.

```
def m_pq() :
```

8.1.6. KNN**8.2. Resultados**