1. Mejora de imágenes y revisión de Python/NumPy

Resumen

En esta primera práctica de laboratorio aprenderemos a modificar los niveles de grises de una imagen con el objetivo principal de mejorar su calidad visual. Esta mejora puede ser de interés para la percepción humana o para procesos computacionales posteriores. También revisaremos la programación en Python y bibliotecas como NumPy, que se usan comúnmente en contextos científicos, y en visión por computadora en particular, ya que las usaremos a lo largo de este curso.

Palabras clave

Aclarar y oscurecer imágenes • Ecualización de histograma • Python • NumPy

Contenido

1. Introducción

Asumimos conocimientos generales y básicos de Python. Por lo tanto, solo mencionaremos algunos conceptos que es bueno revisar y tener en cuenta, principalmente en lo que respecta a cálculos de imágenes y matrices, usando NumPy . En particular. Al menos algunas partes de la programación Array con NumPy puede ser útil.

Fundamentos. Entre las cosas más básicas que debemos saber, necesitamos crear e inicializar matrices, encontrar sus tamaños y realizar operaciones matriciales, tanto por elementos como matriciales . Por ejemplo, si a y b son dos matrices (matrices) 2D del mismo tamaño, ¿qué estamos calculando escribiendo a*b? ¿Es el producto matricial o el producto por elementos? Siguiendo algún tutorial (ejemplo) nos ayudará en estos primeros e importantes pasos. Para cargar y guardar imágenes usaremos Python Imaging Library (PIL). Algún tutorial introductorio. o vídeo corto puede resultar útil. Tenga en cuenta que no cubriremos OpenCV.

Vectorización. En términos generales, debemos evitar escribir bucles explícitos tanto como sea posible, por dos razones: legibilidad y eficiencia. En cuanto a la legibilidad, un código sin bucles tiende a tener menos líneas de código. En cuanto a la eficiencia, las versiones vectorizadas del código se ejecutan significativamente más rápido (las tasas de aceleración pueden ser de uno o incluso más órdenes de magnitud). ¿Cómo podemos vectorizar? Básicamente, mediante el uso de estructuras de sintaxis apropiadas y funciones predefinidas (NumPy) que están vectorizadas y optimizadas (ver ejemplo). Además del código, el problema en sí podría ser vectorizado (ver Vectorización de problemas). pero este es un concepto avanzado que no estamos considerando en nuestro curso. A estas alturas, basta con que te acostumbres a evitar los bucles de escritura (al menos los más obvios) y busques formas vectoriales alternativas de tu código.

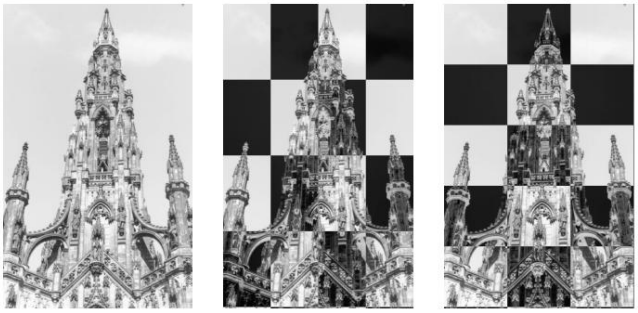
Indexación y corte. Es una habilidad esencial saber cómo acceder a elementos o bloques de elementos dentro de una matriz de dos o más dimensiones. De alguna manera relacionado con esto, muchas funciones de NumPy incluyen un parámetro 'eje' para indicar la dimensión en la que se debe realizar la operación. Aunque se trata de un parámetro opcional y el valor predeterminado puede ser bueno para la operación prevista, no siempre es así. Por ejemplo, uno puede terminar aplicando una función sobre la matriz aplanada 1D cuando esto no es lo que espera. Por lo tanto, es importante conocer o leer atentamente la documentación para asegurarse de que la operación funcione según lo previsto. Realizar pruebas interactivas en la consola de Python, antes de incluir el código en el programa Python que estamos desarrollando, suele ser de gran utilidad

Radiodifusión. En el lenguaje NumPy, radiodifusión se refiere al tratamiento que NymPy hace de las matrices que participan en las expresiones, de modo que sean posibles operaciones que involucren matrices de diferentes tamaños. Por ejemplo, ¿podemos sumar una matriz de 10×8 con otra de tamaño 8×1? O, ¿cómo podemos sumar una constante a todos los elementos de una matriz? Es importante ser conscientes de este concepto para poder utilizarlo como queramos, o entender cuándo se está utilizando, tal vez sin querer. La radiodifusión es una forma de vectorización que, a cambio de la aceleración temporal , puede requerir más memoria, pero esto suele ser menos preocupante a menos que estemos trabajando con muchos arrays al mismo tiempo, o que sean enormes. Algunos esquemas aclaratorios sobre radiodifusión o un vídeo explicativo puede ser útil.

Copiar variables Una posible fuente de problemas o confusión tiene que ver con asignaciones que no corresponden a copias reales, sino a referencias. Por ejemplo, para copiar una matriz NumPy se debe hacer explícitamente (copiar). Una revisión de las declaraciones de asignación en Python puede ayudar a aclarar algunos aspectos relacionados.

2. Aclarar y oscurecer imágenes

Las funciones básicas para aclarar y oscurecer imágenes son sencillas. Compare su expresión matemática con su implementación en el código proporcionado. Compruebe estas funciones comparando ambas



(a) (c) (b)

Figura 1. Visualización del patrón de tablero de ajedrez (Ejercicio 4) aplicado a la imagen original (a) con celdas de 4×4 (b) y 5×3 (c)

las imágenes y sus histogramas antes y después de aplicarlas .

3. Ecualización del histograma

Para comprender mejor la ecualización del histograma y saber cómo implementarla en Python, estamos escribiendo nuestra propia versión de esta operación. Por favor, estudie y verifique el código proporcionado. Como antes, visualice las imágenes y sus histogramas antes y después de la ecualización y asegúrese de comprender el efecto de la ecualización del histograma.

4. Ejercicios

El objetivo principal de los siguientes ejercicios es adquirir mayor fluidez en Python, principalmente en lo que respecta a la manipulación de imágenes y matrices con NumPy. No necesariamente tienes que hacer todos los ejercicios; sólo reflexiona sobre cuánta práctica necesitas para conseguir las habilidades necesarias para resolver, con esfuerzo moderado, ejercicios de este tipo y nivel de dificultad.

1. Resuma los conceptos y las funciones de Python (y NumPy) utilizados en esta práctica de laboratorio que sean (más) novedosos para usted. El propósito es que esto te ayude a comprenderlos e internalizarlos , y también te sirva como referencia posterior. Intente organizar este resumen por categoría (por ejemplo, NumPy, PIL, Matplotlib) y hágalo lo más visual y rápido de consultar posible.

2. En el código proporcionado, guardamos una imagen en el disco usando varias líneas de código. Reemplace esas líneas por una sola con la misma funcionalidad definiendo y llamando a una función saveImg(). Decide qué parámetros debe tener esta función.

3. Generalice la función de brillo, brightenImg() para que funcione tanto con imágenes en nivel de grises como en color.
   Puede tratar cada banda de color como si fuera una única imagen de nivel de grises. No olvide mostrar la imagen resultante y guardarla en el disco. Puedes probar esta función en imágenes que previamente fueron oscurecidas con darkenImg(), lo cual también puedes generalizar.

4. Escriba una función checkBoardImg(im,m,n) que, dada una imagen im en nivel de grises , cree una imagen del mismo tamaño y del mismo contenido, pero que invierta o no los píxeles, alternativamente, siguiendo un patrón de tablero de ajedrez. tern donde la imagen se divide en m × n celdas, como se ilustra en la Fig. 1.

   Luego verifique el correcto funcionamiento de esta función con la ayuda de una función adicional, pruebe CheckBoard() que llama a checkBoardImg() para una de las imágenes disponibles y para un número particular de celdas. Puede utilizar el showInGrid() proporcionado para mostrar los resultados. Reutilice la función saveImg() para guardar la imagen resultante.

5. A menudo resulta conveniente procesar o representar imágenes en forma de árbol de varios niveles. Por ejemplo, el primer nivel considera la imagen completa; el segundo nivel considera las cuatro subimágenes correspondientes a dividir la imagen original en cuatro cuadrantes, y así sucesivamente hasta llegar a un número determinado de niveles.

   Escriba una función multiHist(im,n) que devuelva una lista de histogramas de niveles de grises de la imagen im correspondientes a n niveles. Por tanto, si n = 1, la lista devuelta constará de un único histograma; si n = 2, la lista tendrá 5 (1+4) histogramas; si n = 3, tendrá 21 (1+4+16) histogramas, etc.

   Supongamos que la imagen de entrada es una matriz NumPy 2D (no un objeto Imagen del módulo PIL). Nuevamente, puede usar nuestra función showInGrid() para ayudarlo a desarrollar o probar la función.

   Como ejemplo, con n = 2 y usando histogramas de 3 bins (para simplificar la salida de la consola), el resultado de nuestra implementación de multiHist() en la imagen de la Fig. 1a es

   ```
   [matriz([ 25837, 53944, 193375]), matriz([ 1811, 3782,
     62593]), matriz([ 3835, 7701, 56650]), matriz([ 9006,
     19530, 39856]), matriz([10921, 21955, 35516])]
   ```

6. Escriba una función expTransf(alpha,n,l0,l1,bInc) que genere una función de transformación de nivel de grises T(l) = +b para n valores $^{-\alpha l\, a^2\, e}$ de entrada en el rango [l0,l1]. La salida debe ser una matriz 1D de n valores de nivel de gris, en el mismo rango, aumentando o disminuyendo dependiendo del valor del valor booleano bInc. Establezca a y b para garantizar el rango. Tenga en cuenta que puede calcular la salida asumiendo bInc=True y simplemente invertirla si bInc=False. Muestre estas funciones para diferentes rangos de niveles de gris y valores α > 0 . Espacio de línea de Numpy () La función puede ser útil aquí. Luego, aplique estas transformaciones a las imágenes de entrada im a través de otra función transfImage(im,f) para diferentes funciones f generadas con expTransf().